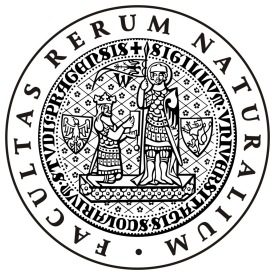
UNIVERZITA KARLOVA V PRAZE

Přírodovědecká fakulta

Katedra aplikované geoinformatiky a kartografie



METODY A NÁSTROJE PRO ANALÝZU PROSTOROVÉ DOSTUPNOSTI V GIS

THE METHODS AND TOOLS FOR SPATIAL ACCESSIBILITY ANALYSIS IN GIS

Bakalářská práce

Tomáš Loukotka

srpen 2008

Vedoucí bakalářské práce: Mgr. Stanislav Grill

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a že jsem všechny použité prameny řádně citoval.

Jsem si vědom toho, že případné použití výsledků, získaných v této práci, mimo Univerzitu Karlovu v Praze je možné pouze po písemném souhlasu této univerzity.

Svoluji k zapůjčení této práce pro studijní účely a souhlasím s tím, aby byla řádně vedena v evidenci vypůjčovatelů.

V Praze dne 21. srpna 2008 …….....

Tomáš Loukotka

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé práce Mgr. Stanislavu Grillovi za věnovaný čas, cenné rady a připomínky ke stěžejním částem práce. V neposlední řadě mu patří též dík za psychickou podporu a motivaci v průběhu sepisování práce.

Metody a nástroje pro analýzu prostorové dostupnosti v GIS

Abstrakt

Cílem práce je srovnání různých přístupů k analýze prostorové dostupnosti za účelem usnadnění rozhodovacího procesu výběru nejlepší metody pro řešení konkrétního problému a výběru konkrétního softwarového nástroje. Jedním ze základních bodů práce bude odlišení možností a využitelnosti analýz prováděných s použitím digitálních dat v rastrovém nebo vektorovém datovém modelu. Zhodnoceny jsou možnosti vybraných softwarových nástrojů. Zvláštní pozornost je věnována extenzím Spatial Analyst a Network Analyst softwaru ArcGIS. Součástí práce je program v jazyce Java, který umožňuje srovnávat algoritmy s různými parametry z hlediska přesnosti výstupních dat a časové náročnosti.

Klíčová slova: prostorová dostupnost, nákladový povrch, síťová analýza, Dijkstrův algoritmus

The methods and tools for spatial accessibility analysis in GIS

Abstract

The goal of this paper is to compare different approaches to spatial accessibility analysis in order to ease the decision making, when choosing the best method for the particular problem and choosing the particular software tool. One of the key component of the work shows the distinction of possibilities and usability of analysis using digital data in either raster or vector data model. The possibilities of chosen software tools are evaluated. The special attention is given to Spatial Analyst and Network analyst extensions in ArcGIS. The paper also contains a program in Java language, which allows the comparison between algorithms with various arguments in respect of output precision and time costingness.

Keywords: spatial accessibility, cost surface, network analysis, Dijkstra's algorithm

OBSAH

SEZNAM OBRÁZKŮ A TABULEK

KAPITOLA 1
Úvod

Prostorová dostupnost a její analýzy se v dnešním světě stávají stále důležitější částí rozhodování o lokalizaci objektů. Slouží také ke zpětnému hodnocení této lokalizace či jako podpůrný aparát pro jiné analýzy v prostoru.

Tato práce si klade za úkol seznámit čtenáře s těmito analýzami a umožnit mu pochopení základních pravidel při jejich uskutečňování a při jejich přípravě. Vzhledem k rozsahu práce se nemůže jednat o seznámení úplné, takže některé aspekty jsou potlačeny či zcela pominuty. I přesto se domnívám, že čtenář po přečtení práce získá alespoň základní povědomí o tom, co to analýzy prostorové dostupnosti jsou a pochopí, jakým způsobem jsou prováděny. Čtenář již seznámený s problematikou zde nalezne srovnání vybraných softwarových nástrojů a také srovnání vybraných algoritmů, které mu umožní, aby sám potřebné softwarové nástroje vytvořil. Cílem je, aby práce sloužila čtenáři jako průvodce ve chvíli, kdy potřebuje použít nějakou analýzu prostorové dostupnosti. Tj., aby mu umožnila lépe se rozhodnout pří výběru správné metody a nástroje.

Obecně tato práce chápe metodu jako postup, kterým ze zadání získat potřebný výsledek. Přitom jsou používány různé nástroje. V případě analýz prostorových dostupností je postup práce následující: Na základě zadání analýzy jsou připravena vstupní data a vybrán nástroj, pomocí kterého na základě těchto dat vznikají data výstupní. Ta jsou pak vizualizována a interpretována. Celým tímto postupem se práce postupně zabývá, důraz je kladen zejména na popis nástrojů. Naopak interpretaci se práce nevěnuje.

První část se věnuje definici termínu prostorová dostupnost a v návaznosti na to též termínu analýza prostorové dostupnosti. V literatuře se můžeme setkat s různě širokým vymezením těchto termínů, a proto je třeba si tuto definici ujasnit. Autor volí spíše užší definici, a to zejména z důvodu rozsahu práce.

Další kapitola pojednává o přípravě dat před tím, než mohou být použita k analýze. Jsou zde zmíněny různé možnosti získání vstupních dat a také popsán rozdíl mezi rastrovými a vektorovými daty a jejich vhodnost či nevhodnost pro různé druhy analýz. Součástí je též stručné shrnutí možností vizualizace.

Další část již popisuje konkrétní softwarové nástroje. Velká pozornost je věnována produktu firmy ESRI ArcGIS, konkrétně rozšířením Spatial Analyst a Network Analyst. Tyto rozšíření představují 2 možné přístupy k problému analýz. Spatial Analyst je řeší nad rastrovými daty a Network Analyst nad vektorovými. Toto je jedna ze základních otázek práce. Nad jakými daty analýzu provádět? Kdy zvolit tento způsob a kdy ten druhý? Krom ArcGISu je věnována pozornost též ILWISu a GRASSu, což jsou nástroje, které mají výhodu v tom, že jsou distribuovány jako OpenSource a jsou pro uživatele k dispozici zdarma. Přes jejich, ve srovnání s ArcGISem, nepříliš přívětivé prostředí ho v některých možnostech dokonce předčí. Na závěr kapitoly se autor pokusí tyto rozdíly popsat. Současně s problematikou výběru nástroje tato část také řeší vizualizaci výsledných dat, která je součástí nástroje, který analýzu provádí.

V další části jsou popsány algoritmy, které se používají pro analýzy prostorové dostupnosti. Tato část je nutným předznamenáním pro srovnání různých algoritmů, či jednoho algoritmu s různými parametry. Nejvíce prostoru je věnováno Dijkstrově algoritmu.

Následuje část, kdy je popisován autorem napsaný program. Nejedná se o kompletní kód, ale jen popis hlavních tříd a metod v jazyce Java a popis možností programu. Kompletní kód je pak součástí přílohy. Program je určen ke srovnávání různých algoritmů. Výsledky tohoto srovnání jsou pak součástí další kapitoly. Program je nedílnou součástí práce a lze ho využít i více, než ho využívá tato práce.

Poslední část práce se věnuje srovnání jednotlivých algoritmů z různých hledisek. Těmito hledisky jsou zejména přesnost výstupních dat a časová náročnost, a to s ohledem jak na zvolená vstupní data, tak na použitý algoritmus či jeho použitou část. V závěru jsou pak všechny tyto poznatky shrnuty.

KAPITOLA 2
Vymezení pojmů

Pro další části práce je potřeba se nejprve seznámit se základními pojmy, které budou použity v práci. Jedná se především o pojmy dostupnost a analýza prostorové dostupnosti. V práci jsou často používány výrazy, které jsou převzaty z běžné terminologie, ale také výrazy, pro něž z anglických výrazů neexistuje jednotný překlad. Proto bude význam těchto pojmů objasněn hned na začátku kapitoly.

2.1 Termíny použité v práci

V této části kapitoly budou zmíněny pojmy hojně používané v práci. Nejedná se samozřejmě o všechny použité odborné pojmy, ale jen ty nejpoužívanější a ty, u kterých by hrozilo nepochopení významu. Nejprve jsou rozebrány vybrané pojmy ze Slovníku VÚGTK, následují vlastní autorovy definice, které se opírají zejména o anglické originály výrazů.

- **hrana** – orientované spojení dvou krajních uzlů
- **uzel** – začátek nebo konec linie, uzel může být sdílen několika liniemi
- **pixel** – nejmenší prvek zobrazovací plochy, jemuž lze nezávisle přiřadit barvu nebo intenzitu (šedi)
- **rastrová data** – prostorová data vyjádřená formou matice buněk nebo pixelů
- **vektorová data** – polohová data ve formě souřadnic konců úseček, bodů, poloh textů apod.
- **vrstva** – sada prostorových dat, týkajících se jednoho tématu nebo majících společný atribut, vztažená k jednotnému souřadnicovému systému

(VÚGTK, 2008 pro všechny výše uvedené výrazy)

Nyní následují pojmy vymezené autorem:

- **výchozí bod** – z angl. origin; bod, ze kterého vychází pohyb
- **cílový bod** – z angl. destination; bod kam směřuje pohyb

- **nákladový povrch** – z angl. cost surface; jedná se o prostorová data v rastrové podobě, kdy každý pixel nabývá hodnoty úměrné tomu, jak velký odpor (angl. impedance) klade jeho plocha, když se přes tuto plochu nějaký subjekt pohybuje. Při analýze nad vektorovými daty se náklady nezískávají z nákladových povrchů, ale z atributů jednotlivých entit vrstvy.
- **obslužné území** – z angl. service area; jedná se o území, které náleží obslužnému zařízení. Pro každý bod v tomto území je toto obslužné zařízení nejdostupnější.
- **obslužné zařízení** – zařízení poskytující služby; znázorněno bodem. Často se jedná o cílový bod pohybu.

2.2 Definice dostupnosti

Definice dostupnosti lze rozdělit do dvou základních skupin.

První skupina chápe dostupnost jako vlastnost místa, která vyjadřuje jeho „blízkost" k ostatním místům a to komplexně (JIANG, CLARAMUNT, BATTY, 1999). Tato skupina může brát do úvahy i atraktivitu místa. Počet výchozích i cílových bodů je v podstatě neomezený. Pro každý bod je na základě atraktivity a vzdálenosti (lépe řečeno obtížnosti tuto vzdálenost překonat) ostatních relevantních lokací možno spočítat jedinečnou hodnotu dostupnosti.

Dle Jianga, Claramunta a Battyho (1999) je dostupnost součtem hodnot, které jsou funkcí nákladů na navštívení určitého místa a prospěchu z této návštěvy. Určení míst pro která se součet provádí pak výrazně analýzu ovlivní. Formalizované vyjádření:

$A_i = f(W_j, d_{ij})$,

kde A_i je komplexní dostupnost lokality i, W_j prospěch, který přináší návštěva lokality j, a d_{ij} je obtížnost k překonání vzdálenosti mezi i a j.

Miller (2000) ve své práci zmiňuje 3 teorie dostupnosti: První teorie se nazývá přístup založený na omezeních a vymezuje všechna místa, která je při daných rychlostech cestování a trvání jednotlivých aktivit možno navštívit. Tato teorie vychází z Hägerstrandova časoprostorového prizmatu. Druhý přístup měří dostupnost v souvislosti s atraktivitou navštíveného místa. Je podobný metodě popsané v předchozím odstavci. Vychází z přístupu Weibulla (1976), který chápe dostupnost jako blízkost z určitého bodu do skupiny cílových bodů, z nichž každý má svojí charakteristickou veličinu – atraktivitu. Třetí systém je založen na měření užitku, který má uživatel když zvolí návštěvu konkrétního místa.

Shen (1998) uvažuje mimo zmíněné definice též dostupnost s použitím různých prostředků dopravy. Nezůstává jen u dopravy osob, ale snaží se tyto teorie aplikovat i v oblasti

telekomunikací a informačních technologií. Důležitá zde není ani tak rychlost přenosu, ta už je v dnešní době zanedbatelná, ale přístup ke službě.

Výše zmíněné modely počítají spíše s dostupností spočtenou pro místa, kde lidé bydlí. Agregátní veličina, pro každé místo jedinečná, by se dala nazvat spíše míra dostupnosti jiných míst. Měří se u objektů, které generují „poptávku". Naproti tomu stojí dostupnost jako agregátní veličina, taktéž pro každé místo jedinečná, která by se dala nazvat jako míra dostupnosti z jiných míst. Tuto dostupnost má smysl měřit u objektů, které generují „nabídku" – nejčastěji služeb(obchody, nemocnice), přírodních zdrojů (studna) a udává míru snadnosti přístupu ke konkrétní službě či zdroji lokalizované v daném objektu (viz dále). Existují samozřejmě též objekty u kterých můžeme měřit obě tyto míry. Příkladem může být hotel, kdy jeho dostupnost směrem k hotelu udává míru, jak je těžké či snadné pro zákazníky se do něj dostat a dostupnost od hotelu udává, jak snadno mohou klienti využívat dalších služeb, které okolí nabízí (např. přístup do centra, přístup k nakupovacím centrům atd.). Tedy např. hotel vybudovaný přímo u letiště bude mít určitě relativně lepší dostupnost směrem k hotelu než dostupnost směrem od hotelu (za předpokladu, že se zaměřuje na zahraniční klientelu). Rozdíl mezi těmito přístupy je především ve směru prostorového pohybu. U prvního počítáme s pohyby vycházejícími z objektu, u druhého s pohyby přicházející k objektu.

Všechny modely založené na dostupnosti jako jedinečné vlastnosti dané lokace narážejí na několik problémů v jejich definici. Je velmi složité určit pro každý objekt relevantní související objekty. Dost modelů též počítá s užitky, které vznikají návštěvou určité lokality. Určení těchto benefitů je problémem spíše sociální geografie či sociologie, proto se jím práce nebude zabývat.

Druhá skupina přístupů chápe dostupnost jako míru, která je podobného druhu jako vzdálenost, lze ji tedy měřit mezi 2 lokalitami. Takto chápaná dostupnost je nepřímo úměrná ceně za změnu lokace (viz výše – aspekt odporu při překonávání vzdálenosti). Zapsat tento vztah se dá takto:

$A_{ij} = f(C)$ (CHAN, 2005) ,

kde A_{ij} je dostupnost lokality j z bodu i, a C jsou náklady na překonání této cesty (viz. předchozí odstavce).

Z výše uvedených odstavců je patrné, že je třeba si vymezit dostupnost tak, jak bude chápána v této práci. Bude zařaditelná do druhé jmenované skupiny, tj. dostupnost jako vlastnost dvou bodů (i s orientací).

Dostupnost tedy tato práce vnímá jako specifickou vlastnost dvojice objektů (z nichž z jednoho pohyb vychází a do druhého směřuje), mezi kterými probíhá prostorový pohyb subjektů, a tato vlastnost závisí na nákladech, které jsou potřeba k překonání této vzdálenosti v širším slova smyslu. Přístupy jiných autorů k této definici mohou být nalezeny v publikaci Accessibility Analysis Literature Review (HALDEN; JONES; WIXLEY, 2005). Poněkud problematické se v tomto světle jeví definování objektu. Pro potřeby práce budeme objekt

aproximovat bodem či pixelem. Pohyb tedy budeme chápat tak, že probíhá z bodu do bodu, či z pixelu do pixelu.

2.3 Definice analýzy prostorové dostupnosti

Pro potřeby práce je třeba si detailněji objasnit termín analýza prostorové dostupnosti. Za analýzu prostorové dostupnosti lze považovat takovou analýzu nad prostorovými daty, která se zabývá dostupností. (viz. předchozí část kapitoly). Pro potřeby práce bude na konci kapitoly tato definice zpřesněna.

V literatuře se lze setkat s různě širokými vymezeními toho, co je to analýza prostorové dostupnosti. V této práci bude uvažován jako subjekt vykonávající pohyb v prostoru pouze někdo s vlastní rozhodovací schopností (nejčastěji tedy člověk, ale samozřejmě by se teoreticky mohlo jednat např. o zvíře). Tento subjekt má možnost si volit cesty, a analýza prostorové dostupnosti mu má v tomto rozhodování pomoci. Občas bývá do těchto analýz zahrnovaná i rozsáhlá skupina analýz, vykonávaných na subjektech, jejichž pohyb je přímo dán vlastnostmi prostředí (typicky např. voda, jejíž pohyb v terénu je dán jeho sklonem). U těchto subjektů lze provádět např. drenážní operace ad. Práce se jimi nebude zabývat z následujících důvodů: U těchto sítí mají spíše smysl otázky: Kudy se bude pohybovat objekt z tohoto místa? Může dosáhnout mnou specifikovaného místa? Mezi dvěma body tedy nemá většinou smysl měřit dostupnost, ale nejprve zjistit, jak bude z daného výchozího bodu pohyb vypadat. Vstupní data zde nemají podobu nákladových povrchů, slouží k tomu, abychom v každém místě mohli odvodit jak se bude objekt dále pohybovat (typicky např. DEM). Do trochu jiné kategorie pak spadají také různé analýzy toku v sítích. Ani zde není primárním problémem dostupnost mezi dvěma body. Proto se práce bude zabývat takovými analýzami, kdy mezi 2 body lze spočítat jejich dostupnost (v obou směrech). A s takto naměřenou dostupností analýzy pak dále budou pracovat.

Postup při analýzách prostorové dostupnosti lze rozdělit do 4 kroků (LIU; ZHU, 2004): Prvním z nich je stanovení cílů analýzy a formulace základních konceptů k měření dostupnosti. V druhém kroku následuje formulace toho, jak budeme dostupnost měřit, a jaké aspekty zahrneme do analýzy (toto téma podrobněji v předchozích odstavcích). Tato formulace vychází jak z prvního kroku, tak i z dostupných dat, charakteru území apod. Následuje spočtení dostupnosti (dostupností) dle zadaných konceptů (což většinou zahrnuje zejména spočtení nákladů, viz dále) a kalibrace veličiny, kterou je měřena dostupnost (viz. předposlední odstavec kapitoly). Následuje vizualizace a interpretace výsledků.

Do analýzy můžeme dle Liu a Zhu (2004) zahrnout následujících 7 aspektů analýzy. Definice toho, co to dostupnost je, se pak liší v závislosti na tom, jak tyto aspekty specifikujeme a také na tom, jestli je vůbec specifikujeme (některé z nich). Jak uvidíme, nejedná se o aspekty na sobě nezávislé, ale vzájemně se ovlivňující.

Prvním z nich je specifikace prostorových jednotek, ve kterých bude analýza probíhat. Pod tímto je chápáno určení, jaké prostorové entity budeme uvažovat jako jednotku. Může se jednat o bod (což je v reálných případech těžko uvažovatelné), linii , polygon (může se jednat o plochu na které se nachází obchod, ale i o blok budov, čtvrť, město, atd.), teoreticky lze uvažovat i 3D objekty a chápat prostorový pohyb trojrozměrně. Práce bude pracovat s jakýmikoliv prostorovými jednotkami, které jsou převedeny na bod, linii či pixel.

Dále je třeba specifikovat socio-ekonomické skupiny. V obecnější rovině je třeba specifikovat podmnožinu objektů, které vykonávají pohyb v prostoru. Jedná se tedy vlastně o přesné vymezení objektů, u kterých budeme zkoumat prostorový pohyb. Tento aspekt ovlivňuje atraktivitu (kterou se práce nezabývá) a může ovlivňovat také způsob pohybu. Tuto závislost však v práci v úvahu brát nebudu.

Dalším aspektem specifikace příležitostí, tedy objektů, které jsou z nějakých důvodů cílem prostorového pohybu, toto je vlastně bližší specifikací prvního aspektu a to prostorových jednotek, které se analýzy účastní.

Čtvrtým aspektem je specifikace způsobu pohybu. Tento aspekt přináší možnost zahrnout do analýzy více druhů pohybu (např. cestování pěšky, autobusem, tramvají, autem,) a tyto pak mezi sebou srovnávat či tyto různé skupiny agregovat a provádět tak komplexnější analýzu. Tento aspekt ovlivňuje tvorbu nákladových povrchů.

Definice výchozích a cílových bodů pro pohyb v prostoru je vlastně shrnutí prvních třech aspektů (tedy toho, jak velké jednotky a jaké povahy se analýzy účastní, za jakým účelem je prováděn pohyb a kdo se ho účastní) do jednoho s tím, že může počet takto definovaných objektů ještě zúžit na konkrétní výchozí a cílové body (např. na jedno místo, jež je cílem prostorového pohybu z relevantních míst).

Atraktivita přidává do analýzy další prvek, kdy cílovým místům přiřazuje určitou hodnotu a na základě té, je ovlivněno rozhodování (či nutnost) subjektů vykonávajících prostorový pohyb při volbě cílů pro svůj pohyb. Práce se tímto aspektem zabývat nebude.

Konečně odpor při konání prostorového pohybu určuje snadnost, s jakou je pohyb mezi určitými dvěma místy vykonáván. Chan (2005) vyjadřuje náklady na překonání vzdálenosti mezi 2 body následovně:

$$C = w_1.c_t + w_2.c_d + w_3.c_c + c_{o,}$$

kde c_t jsou náklady spojené s vynaloženým časem, c_d náklady spojené se vzdáleností, c_c hotovostní náklady a c_o ostatní náklady(jako např. nepohodlí atd.). w_{1-3} jsou pak příslušné váhy. Toto nejobecnější vyjádření lze samozřejmě libovolně zjednodušit tak, že vezmeme v úvahu pouze vstupní data udávající jen jeden typ nákladů (nejběžněji časové náklady).

Analýzy dostupnosti se liší v závislosti na tom, jak a jaké jsou specifikovány aspekty (viz. výše – 7 aspektů). Lze je rozdělit do 2 základních skupin a to problémy formulované otázkami:

„Co když umístím objekt na toto místo?“ a „Kam mám umístit objekt?“ (VAN ECK, 1999) V prvním případě máme jasně definovaný výchozí bod (body) ze kterých (nebo do kterých) měřím, v druhém případě jsou pro mě potenciálními výchozími body všechny lokality zahrnuté do analýzy.

První skupina (formulovaná otázkou: „Co když umístím objekt na toto místo?“) zahrnuje problémy kdy známe cílový bod či body, a postupným počítáním dostupností z výchozích bodů po celém území získáváme informace o tomto území. Postup může být i opačný, tj. z výchozích bodů se počítá dostupnost do bodů v terénu. Tento druhý způsob je nadále v práci zmiňován. Lze si ho však bez problémů představit jako způsob první (samozřejmě v případě nákladů, které se liší podle směru pak analýza přináší jiné výsledky). Tyto informace mohou např. udávat vzdálenost od objektu či náklady na pohyb k němu, vymezení obslužných území, vymezení směrů pohybu, různé typy kvantifikací v rámci obslužného území atd. (např. počet obyvatel území). Základem těchto analýz je vždy spočtení kumulovaných nákladů pro překonání dané vzdálenost. Dále se práce bude zabývat problémy, kdy známe výchozí i cílový bod (popř. více bodů s určením či bez určení pořadí) a cílem je nalézt nejoptimálnější cestu. Zmíním i Steinerův strom, ačkoliv se nejedná o problém, kde by šlo o minimalizaci nákladů při pohybu, ale o minimalizaci nákladů při tvorbě sítě.

Druhou skupinu lze pojmenovat jako problémy umístění. Jsou to problémy kdy nejsou známy počáteční či koncové body a je snaha je najít. Může jít o problémy následujícího druhu: vytvořit oblasti se stejným počtem měřitelné jednotky (obyvatel, poptávky ad.), vytvořit oblast, ke které náleží veličina o určité minimální velikosti, a naopak nepřekračuje nějakou hodnotu ad. Práce se jimi zabývat nebude.

Poté, co je nadefinována veličina dostupnost a změřena přichází na řadu kalibrace výsledků. Tento proces spočívá, v úpravě vstupních parametrů analýzy za účelem dosažení věrnějšího odrazu reality. Zejména se mohou měnit váhy položené na jednotlivé druhy nákladů.

U vizualizace dat budou popsány různé způsoby jak ji provádět a to v kontextu hodnocených GIS. Mimo to se v práci nachází stručné popsání těchto postupů, které je zobecněním postupů, které jsou použity v GIS, či které autor považuje za použitelné.

Tato kapitola řešila následující problém: Vymezit pojem analýza prostorové dostupnosti a stanovit, kterými druhy těchto analýz se bude práce zabývat. Výsledkem je následující vymezení: Práce se bude zabývat analýzami, ve kterých má smysl počítat dostupnost (náklady na pohyb) mezi 2 body a tento pohyb vykonává subjekt, který má možnost se rozhodovat.

KAPITOLA 3
Příprava analýz

Tato kapitola se zabývá úkony, které je nutné provést před tím než může dojít přímo k vytvoření výstupních dat ze vstupních dat pomocí konkrétního nástroje. Zároveň jsou zde i zmíněny možnosti vizualizace, která sice probíhá až po vytvoření výstupních dat, ale většinou se způsob vizualizace připravuje již předem.

3.1 Vstupní a výstupní data

Při analýze prostorové dostupnosti je potřeba si nejprve ujasnit, proč ji budeme provádět. Tedy kdo bude uživatelem, a jaké požadavky na takovou analýzu má.

Z tohoto zjištění vycházejí požadavky na výstupní data. Ty jsou stručně shrnuty v následujících bodech:

- vymezení území
- prostorová jednotka
- nákladová jednotka
- hodnotová přesnost
- typ analýzy

Tyto body ukazují požadavky, které je třeba si určit, nejlépe v uvedeném pořadí. Nejprve je třeba si vymezit území, na kterém bude analýza probíhat. Zde je potřeba upozornit na rozdíl mezi územím, které vstupuje do analýzy, a územím, které je výsledkem analýzy. Výsledné území musí být vždy menší, či stejně velké jako území, které do analýzy vstupuje. Prostorová jednotka je jednotka, o které potřebujeme mít informace. Stačí nám informace o jednotlivých městech? O jednotlivých zastávkách autobusu? O místech podél jakékoliv silnice? Nebo potřebujeme mít informaci o každé části území? V případě, že ano, jak velká pak má taková část být? Určení nákladové jednotky znamená, že je třeba zvážit, zda bude pro analýzu rozhodující čas, vzdálenost, peněžní náklady, případně nějaká jednotka nákladů sestavená z předchozích jmenovaných. Dále je třeba určit typ analýzy. To spočívá v určení toho, jestli nás budou zajímat náklady, či spíše jen fakt, ke kterému obslužnému zařízení různá místa spadají, případně jiný,

specifičtější typ analýzy. Konečně hodnotová přesnost udává, jak přesná potřebujeme výsledná data. Toto má smysl určovat až po tom, co je určeno, že nás budou zajímat náklady jako takové,a nikoliv jiné informace. Pod hodnotovou přesností si lze představit např. to, že dojezdovou vzdálenost budeme potřebovat s přesností na minuty, případně s přesností na sekundy apod.

Po určení požadavků na výstupní data následuje zjištění požadavků na vstupní data, které je nutno vykonávat současně s výběrem vhodného softwarového nástroje, případně s konstatováním, že tento nástroj bude nutné vytvořit. Vstupní data totiž ovlivňují volbu vhodného nástroje. Nástroj, pro který se nedají sehnat data, byť se jeví jako nejvhodnější, je pro analýzu nevhodným. A naopak jsou-li již zvolena vstupní data, může to nežádoucím způsobem zužovat výběr vhodného nástroje.

Upřesnit si požadavky na vstupní data je nutné v následujících ohledech:

- typ
- jednotky
- hodnotová přesnost
- velikost území

Nejdůležitější u vstupních dat je určit jejich typ, tj. jestli půjde o data rastrová či vektorová. To je značně ovlivněno tím, za jaké prostorové jednotky chceme získávat výstupní data. Pokud je chceme z každé části území je volba rastrových dat logická. Naopak pokud nás zajímají pouze určité objekty, jako třeba města, silnice apod. volíme vektorová data. Důležité je též určit jednotky ve kterých vstupní data budou, konkrétně tedy jejich atribut či hodnota, která se označuje jako náklady na procházení a ve výsledných datech pak určuje hodnotu kumulovaných nákladů. Pak ještě určíme hodnotovou přesnost, ve které tato data budou. Mohou to být minuty, kilometry nebo i jiné složitější ukazatele. Vždy platí, že čím větší hodnota tím obtížnější průchod. Na závěr určíme velikost území na kterém bude analýza probíhat. Ta vyplývá již z charakteru zadání analýzy. Pokud se jedná o analýzu nad rastrovými daty, bývá vhodné území upravit na obdélník, resp. je to jednodušší. Zejména u analýz nad rastrovými daty je potřeba, aby velikost vstupních dat (vstupního rastru) nebyla příliš veliká. Pokud tedy požadavky na prostorovou přesnost výstupních dat nejsou tak velké jako je přesnost vstupních dat, je na místě jejich zmenšení. Vždy je třeba hledat kompromis mezi tím, co je k dispozici a tím, co se jeví jako optimální.

Stavba rastrových dat je jednodušší než stavba vektorových dat. Jedná se o dvourozměrné pole pixelů. Žádné další vazby není třeba mezi pixely dělat. Každý pixel je jednoznačně určen svými souřadnicemi x a y, které jednoznačně určují jeho pozici v systému i pixely jemu sousední. Z tohoto důvodu se v rastrových datech nedají modelovat mimoúrovňová křížení. Hůře se zde modelují i úrovňová křížení, protože vyvstává problém, kterou hodnotou nahradit

pixel, ve kterém se linie kříží. Obecně je rastr tedy vhodnější k modelování dostupnosti s náklady, které jsou vázány spíše na plochu, než na linii. Teoreticky může existovat i rastr, který navíc bude mít vazby, které budou obdobou hrany, tedy vazeb mezi 2 uzly, u vektorových dat. Jednalo by se o jakousi kombinaci obou přístupů, přístup hybridní. Tento přístup čerpá výhody z výhod rastrového i vektorového modelu. Nevýhodou je vyšší náročnost jak programátorská, tak výpočetní.

Vektorové nákladové povrchy se skládají primárně z uzlů a hran. Každý uzel v sobě nese informaci o všech hranách, které jsou na něj napojeny a každá hrana nese informaci o 2 uzlech, které tvoří její koncové body. Takto snadno mohou vznikat mimoúrovňová křížení (tj. křížení kde v místě překřížení hran chybí uzel). Pokud budeme klást důraz i na směr pohybu stačí brát v úvahu pořadí 2 uzlů přidělených hranám a uzlům přiřazovat jen hrany vycházející z něj. Poněkud složitější situace nastává pokud budeme chtít omezit možnost na jednotlivých uzlech libovolně měnit směr. V tom případě je každé hrané navíc třeba přidat seznam hran, přes které lze pokračovat dále. Nákladové atributy mohou být přiřazeny jak hranám tak uzlům, ve složitějším případě i jednotlivým zatočením.

Pokud bychom měli předchozí informace shrnout, dojdeme k následujícím závěrům. Vektorová data nalézají uplatnění při modelování pohybu, který probíhá po předem daných liniích a kde jednotlivé objekty (zástavky, obchody, čtvrti) lze vyjádřit bodem. Výhodou je možnost vyjádřit mimoúrovňová křížení, omezení zatáčení ad. Rastrová data se uplatní při modelování pohybu po ploše, který probíhá všemi směry (HUSDAL, 1999). Výhodou je fakt, že lze získat informaci o každé části povrchu. Také je výhodou, že lze kombinovat více těchto povrchů do jedné analýzy. Nevýhodou je, že nelze takto tvořit složitější modely a obecně jsou tyto analýzy výpočetně náročnější. V úvahu připadá i použití hybridních dat, která se použijí tam, kde potřebujeme informaci o každé části povrchu a zároveň chceme aby pohyb mohl probíhat i po čistě liniových prvcích (např. nákladový povrch znázorňuje krajinu, vektory znázorňují tunely. Pohyb je možný jak pod povrchem přes vektor, tak po povrchu a to přes rastr.).

Často při analýze dochází k situaci kdy jsou potřeba jako výstupní data jiná data než jsou k dispozici jako vstupní data. Pak přichází v úvahu konverze dat. V případě, že má uživatel k dispozici vektorová data, je možná jejich konverze na rastry (možnosti této konverze samozřejmě závisí na použitém GIS (viz. Tab. 1)). V případě rastrových dat může uživatel data digitalizovat do vektorové podoby. To lze však jen v případě menších území, či za použití kvalitních nástrojů na konverzi. Ty již nejsou běžnou součástí hodnocených GIS. I pokud jsou takové nástroje k dispozici, není konverze vždy možná či žádoucí.

Rastry jako vstupní data – tedy nákladové povrchy lze získat různými způsoby. První možností je rasterizace, kdy z linií či polygonů vzniká rastr. Další možností je interpolace na základě vektorové vrstvy obsahující body. Dále je možné získat rastr překlasifikováním, či převzorkováním jiného rastru.

Tab. 1 Převod z vektorového datového formátu do rastrového formátu

Původní data	**Použité funkce**
vektory – linie, polygony	rasterizace
vektory – body	interpolace
rastr	překlasifikování, převzorkování

Pokud jsou vstupní data připravena a jasně stanoveny požadavky na data výstupní, je třeba vybrat metodu a vizualizovat výsledky tak, aby byl splněn účel analýzy.

3.2 Vizualizace výsledků

Vizualizace bývá nedílnou součástí každé analýzy. Samozřejmě je možné, že zadavatel požaduje pouze výstupní data v „syrové“ podobě, např. pro použití v dalších analýzách. Pak se vizualizace neprovádí. Tab. 2 zobrazuje možné přístupy k vizualizaci dat vzniklých analýzou dostupnosti, konkrétně k vizualizaci akumulovaných nákladů.

Tab. 2 Možnosti vizualizace výsledných dat

rastrová data	**vektorová data**
textové pole	textový popis uzlů
pole pixelů - intervaly	textový popis hran
pole pixelů - odstíny	obarvené hrany - intervaly
izolinie	obarvené hrany - odstíny
	rozdělené linie
	izolinie či plochy

U rastrových dat přichází v úvahu textové pole. Tato metoda byla použita v programu, který je součástí této práce. Její výhodou je nejsnazší implementace a také dohledatelnost dat na první pohled. Nevýhody se objevují při vizualizaci větších rastrů. Takový rastr se pak stává velice nepřehledným. V profesionálních programech se nepoužívá, a lze ji využít spíše ke zkušebním účelům. Standardně se používá barevné pole pixelů, a to buď v různých barvách odstupňovaných podle zlomových hodnot, či v plynulých přechodech. První jmenovaná metoda má výhodu, že se z ní snadno čte informace – tento bod již leží za zlomovou hodnotou, zatímco tento bod ne. Druhá jmenovaná, oproti tomu, umožňuje čtenáři získat přesnější pohled na skladbu akumulovaných nákladů, ovšem konkrétní hodnoty jsou hůře čitelné. Často se využívá kombinace barev odstupňovaných dle intervalů s tím, že každý pixel doplňuje textová informace o hodnotě. Tato informace je uživateli zobrazena až ve chvíli, kdy si ji vyžádá.

U vektorových dat většinou jako podklad pro akumulované náklady slouží síť, z níž se tyto náklady počítají. Textem s hodnotou nákladů mohou být označeny uzly, což je přirozenější,

nebo také hrany. Zde vyvstává problém, jestli hraně přidělit hodnotu uzlu následujícího nebo předchozího. Je to spíše teoretická možnost vizualizace. Více se používají hrany obarvené buď několika barvami v různých intervalech nebo v plynulých přechodech. U první možnosti je opět problém s tím, jaké hodnoty hraně přiřadit, druhá možnost se zas jeví poněkud nepřehledná. Další možností je některé hrany rozdělit, pokud se na nich nachází zlomový bod a část hrany obarvit jednou barvou a část obarvit druhou. Tento způsob se často používá, ale je náročnější na programování. Barvu lze ve všech těchto analýzách nad vektorovými daty nahradit tloušťkou linie a to opět intervalově či plynule. Na základě zlomových bodů na hranách lze též zkonstruovat izolinie či izoplochy.

Výše zmíněné způsoby jen nastiňují celou problematiku vizualizace. V zásadě platí, že uživatel či programátor si sám může zvolit způsob vizualizace, který je vhodný pro danou analýzu. Příklady jednotlivých druhů lze vidět v následující kapitole na ukázce konkrétních nástrojů GIS, případně po spuštění programu Analýzy prostorové dostupnosti.

KAPITOLA 4

Softwarové nástroje pro provádění analýz prostorové dostupnosti

V této kapitole bude věnován prostor nástrojům pro provádění analýz. Tyto nástroje jsou zde popsány a na základě jejich funkcionality pak srovnány. Největší prostor je věnován softwaru společnosti ESRI ArcGIS, který je jedním z nejrozšířenějších a nejpoužívanějších produktů. Důležitou součástí popisu je srovnání nástroje Network Analyst s nástrojem Spatial Analyst, konkrétně s funkcemi spadajícími do složky Distance. Na tomto místě je třeba zmínit, že se práce nebude věnovat komerčním konkurentům ArcGISU na poli analýz prostorové dostupnosti a to zejména systému MapInfo AnySite. Důvodem je jejich špatná dostupnost a také daný rozsah práce, který by tímto byl výrazně překročen. Následovat bude stručný popis komplexních systémů ILWIS a GRASS, jež obsahují menší množství funkcí, ale jedná se o volně dostupný software.

4.1 Řešení analýz v prostředí ArcGIS

Pro analýzy prostorové dostupnosti v ArcGIS lze použít dva nástroje, mezi jejichž použitím je značný rozdíl. Jsou to Spatial Analyst a Network Analyst. Zatímco Spatial Analyst pracuje se vstupními daty v rastrové podobě, Network Analyst potřebuje data vektorová. U obou těchto nástrojů je potřeba vstupní data předpřipravit. O způsobech jakými to provést bude řeč v popisu obou nástrojů.

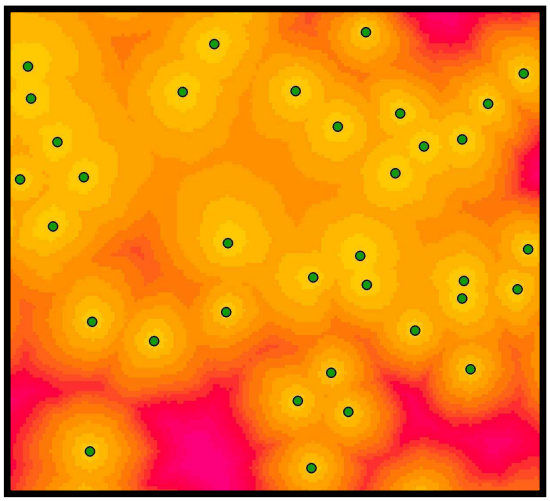
4.1.1 Spatial Analyst

Spatial Analyst zahrnuje řadu nástrojů, z nichž pro analýzu dostupnosti jsou použitelné pouze některé. V tomto odstavci se budu zabývat pouze nástroji souvisejícími se vzdáleností, tedy nástrojů ze skupiny Distance. Do analýz prostorové dostupnosti by se daly zahrnout i skupiny Hydrology, případně Groundwater, ale to by již přesahovalo předem daný rámec práce (viz. definice analýz dostupnosti v kapitole 2).

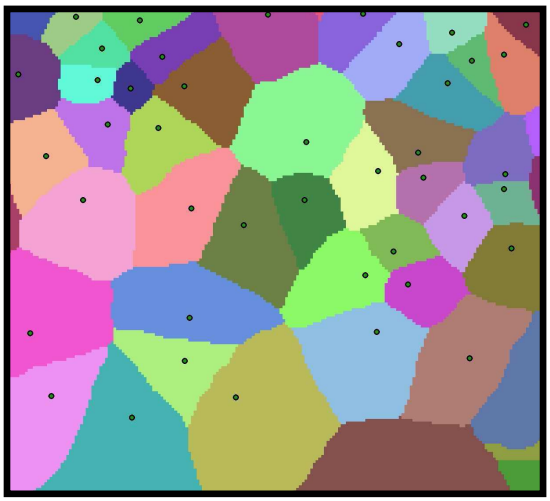
Funkce skupiny Distance lze rozdělit zhruba do 3 skupin, a to hned dvakrát – dle přístupu ke vzdálenosti a dle typu výstupu. Dle přístupu ke vzdálenosti můžeme rozlišit 3 funkce, které používají euklidovský prostor, 3 funkce, které používají pouze „průchozí náklady“ a

nejkomplexnější 3 funkce, které kromě „nákladů na pohyb“ uvažují i terén, možnosti zatáčení atd. Z hlediska výstupu lze funkce rozdělit na alokační, které každému pixelu přiřazují hodnotu, lokality, která je mu nejblíže; měřící pro každý pixel vzdálenost od lokalit; a ukazující směr pohybu při cestě k nejbližší lokalitě.

- **Cost Allocation -** pro vstupní data – obslužná zařízení (vektorová – body, linie; rastrová – všechny pixely, které nenabývají hodnoty NoData) hledá oblast, kterou budou obsluhovat na základě nákladového povrchu. Ve výsledném rastru pak každý pixel nabývá hodnoty, která odpovídá jemu příslušnému centru (ze vstupních dat se volí ještě datový sloupec, který poslouží jako podklad pro toto rozdělení, např. číslo objektu. Všechny pixely, které jsou nejblíže např. objektu číslo jedna, nabývají hodnoty 1.). Lze vytvořit i rastr, kam se uloží vzdálenost od nejbližšího objektu či kalkulovat se vstupním rastrem, kde už jsou území vymezená, a toto vymezení má přednost před počítaným vymezením obslužných území. Další možností výstupu je tzv. back-link rastr, který udává směr k nejbližšímu objektu. Tento směr je vyjádřen jednou z osmi hodnot (nahoru, vlevo-nahoru, vpravo-dolů, atd.) a představuje první pohyb, který se za nejbližším objektem z daného místa uskuteční. Posledním z možných omezení je zadat maximální vzdálenost od objektu, pro vzdálenější území pak již kalkulace neprobíhá.



Obr. 1 Kumulované náklady
(zdroj: ESRI ArcMap 9.1)

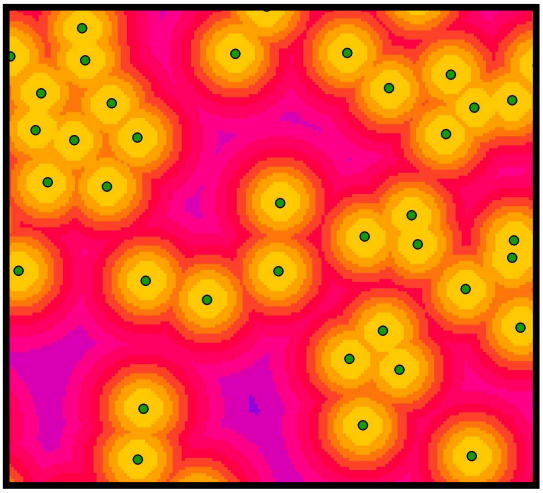


Obr. 2 Obslužná území
(zdroj: ESRI ArcMap 9.1)

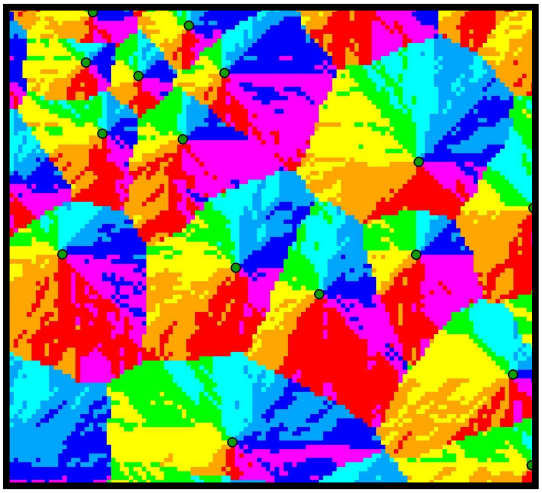
- **Cost Distance a Cost Backlink** – specializovanější nástroje, všechnu jejich funkcionalitu zvládá nástroj Cost allocation(viz výše), který je komplexní.

- **Euclidean Allocation, Distance and Direction** – jsou to nástroje analogické ke Cost Allocation, Cost Distance a Cost Backlink. Pracuje se v euklidovském prostoru, tzn. se berou v úvahu jen „skutečné“ prostorové vzdálenost, do úvahy není brán žádný nákladový povrch. Pro Cost Distance vycházejí okolo objektů kruhy. Euclidean Direction se nemusí omezit jen na 8

směrů, protože směr k nejbližšímu cíli je stálý a lze vyjádřit úhlem. To je jediný podstatnější rozdíl od nástroje Cost Backlink.



Obr. 3 Euklidovská vzdálenost
(zdroj: ESRI ArcMap 9.1)



Obr. 4 Back-link raster
(zdroj: ESRI ArcMap 9.1)

- **Path Distance, Path Distance Allocation, Path Distance Back Link** – jsou komplexnějším rozšířením první skupiny nástrojů. K nákladovému povrchu lze přidat ještě DEM, který mění vzdálenosti, tím že prostor obohacuje o další rozměr (tj. cesta, která by vzdušnou čarou měřila 6 km, může přes kopec měřit 8 km). Dále lze přidat náklady na vertikální pohyb (nejenže přes kopec je vzdálenost delší, ale do kopce pravděpodobně půjdeme pomaleji). Zde můžeme určit faktor, který bude určovat náročnost na procházení terénem v různých úhlech. Faktor může být binární, tj. v určeném intervalu úhlů procházíme bez omezení, mimo interval projít nelze. Dále může být lineární či inverzní lineární, kdy se stoupajícím(klesajícím) úhlem stoupá náročnost. Tyto faktory mohou být ještě symetrické, tj. že se bere v úvahu absolutní hodnota úhlu (tato možnost je docela blízko realitě, kdy jak s velkým stoupáním tak s velkým klesáním je spojena vyšší náročnost na pohyb). Dále může být náročnost průchodu dána funkcí cos, či její inverzní hodnotu, či jejich kombinací. Poslední možností je načtení těchto hodnot pro určité úhly z tabulky. Pro horizontální pohyb lze též definovat faktor náročnosti, a to z hlediska měnění směrů. První z možností je nastavit faktor binární, tj. nad určitý úhel stanovit pixel jako neprůchozí. Další možností (forward) je povolit zatáčení do 45 stupňů, mezi 45-90 stupni aplikovat jistou náročnost a nad 90 stupňů je průchod označen jako nemožný. Dále pak lze nastavit lineární faktor či načtení dat z tabulky.

4.1.2 Network Analyst

Pro provádění analýz pomocí Network Analystu je nejprve nutné ustavit síť (viz. popis vektorového modelu v předcházející kapitole). Běžně jsou modely tvořeny hranami a uzly, které tento model spojují, tak aby na sebe hrany navazovaly a daly se použít pro síťové analýzy. Je

též možné do analýzy zahrnout třetí skupinu a to skupinu, která definuje zatáčení (turns). Při tvorbě sítě je nutno projít následujícími nastaveními.

- **connectivity** – V této části tvorby sítě jsou určeny zdroje (tedy vektorová data) ze kterých bude síť sestávat. Pro tvorbu sítě je možno použít pouze liniová a bodová data. U liniových prvků je nutno určit jestli se mohou spojovat pouze na koncích jednotlivých linií (jednotlivých objektů vrstvy) – nastavení endpoint, nebo v jakémkoliv uzlu dané linie (any vertex), u bodů existuje volba mezi honor a override (override umožňuje navazování bodů do sítě, i jinde než to stanoví volba any vertex nebo endpoint). Dále je možno nastavit více skupin (Connectivity groups), v rámci kterých budou objekty propojeny. Jednotlivé vrstvy mohou být i členy více skupin. Takové vrstvy (většinou bodové) slouží k propojování ostatních vrstev (např. vchody do metra slouží k propojení podzemní sítě s pozemní).

- **elevation** – Zde lze nastavit zdroj výškových dat a na základě tohoto zdroje rozhodnout, zda se objekty propojí či ne. Toto je nutno zjistit pro oba směry (u linie) (tedy pro její začátek i konec)

- **turns** – Do modelu lze přidat 3. skupinu a to turns. Zde lze definovat pro jednotlivé hrany možnosti přechodu na jiné hrany, a to v obou směrech. Je třeba vytvořit shapefile typu turn, přidat do sítě a pak editovat.

- **atributy** – Důležitou součástí sítě jsou atributy, které jsou pak používány v síťové analýze. V ArcGIS existují 4 druhy atributů:
  - **cost –** atribut, který každou hranu či uzel ohodnocuje z hlediska ceny na procházení. Tato cena může být udána v časových jednotkách, či jakýchkoliv jiných nákladových jednotkách. Podstatná je zde ta vlastnost, že při procházení pouze části hrany jsou vynaloženy pouze poměrné náklady. Každá síť musí obsahovat alespoň jeden tento atribut, aby se na ní daly provádět analýzy.
  - **restrictions –** atribut typu boolean, pro každou hranu či uzel určuje, zda je průchozí či ne.
  - **descriptions –** atribut libovolného číselného typu je podobný cost, ale popisuje hranu či uzel z jiného hlediska. Udává jeho vlastnost, která je však po celé délce neměnná, tudíž procházením pouze části hrany tento atribut nabývá stále stejné hodnoty. Může se jednat např. o maximální rychlost, počet pruhů apod. Často se používá pro výpočet nákladových atributů.
  - **hierarchy –** umožňuje nastavit hierarchii jednotlivých objektů pro analýzu. ArcGIS rozlišuje 3 druhy – přibližně v tomto sledu: státní silnice, hlavní silnice, vedlejší silnice. V rámci každé analýzy pak může uživatel stanovit, jaké má preference při používání jednotlivých druhů.

- **directions –** umožňuje pracovat s textovým výstupem. Na základě délky liniových atributů a jejich názvů je schopen generovat „návod“ jak síť procházet. Např.: Po 50 metrech odboč vlevo na King’s Street, tj. funkce, která je podobná té v GPS.

Základem pro analýzy v Network Analystu je definovaná síť (viz. předchozí odstavec) a tzv. síťová místa (network locations), pro která se pak provádějí analýzy. Tyto místa představují výchozí nebo cílové, případně průchozí body. Musí přiléhat na nějaký objekt sítě. Na jaký, to lze nastavit – včetně toho jestli místo může přiléhat na prostředek, okraj hrany či po celé její délce. Lze též nastavit snapping tolerance. Místa lze zadávat buď ručně či pomocí adres apod. (k tomu samozřejmě musí být síť uzpůsobena).

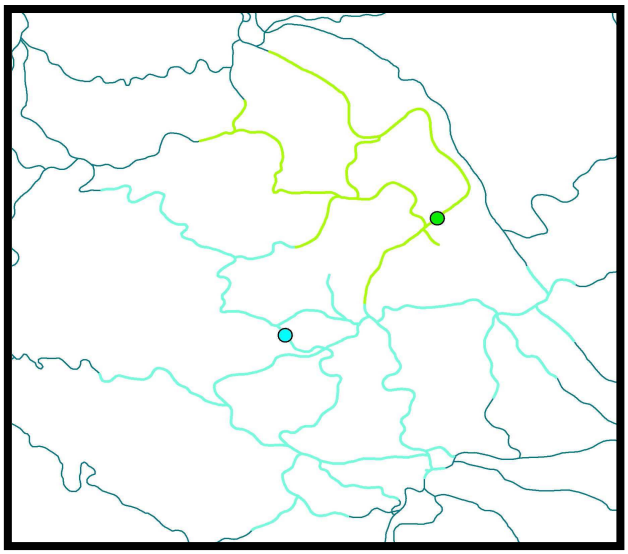
V Network Analystu lze provádět 4 druhy analýz, z nichž každá disponuje mnohými nastaveními. Po zapnutí konkrétní analýzy se objeví vrstva, ve které jako podsložky figurují síťová místa a výsledek analýzy – viz. bližší popis analýz. Druhy analýz jsou následující: hledání nejvhodnější cesty, hledání obslužného území, hledání nejbližší jednotky a vytváření OD cost matice. Tyto druhy analýz nyní budou podrobněji popsány:

- **Route -** Tato analýza vytváří ze zadaných bodů na základě zadaného nákladového atributu nejvýhodnější cestu. Možnosti nastavení jsou následující: lze upravit pořadí procházených míst a optimalizovat ho, přičemž lze nastavit, že první či poslední zastávka bude pevně daná. Dále lze povolit či zakázat tzv. U-turns – tedy obraty na „hraně“, nastavit typ výstupu (skutečná cesta, spojnice bodů, žádný výstup), nastavení hierarchie, možnost počítat akumulovanou hodnotu pro nákladový atribut (pokud je nákladový atribut jen jeden, není třeba toto okno zaškrtávat) a další. Místa pro analýzu udávají buď místa, kterými se bude procházet, nebo mohou tvořit bariéry, tedy místa, přes které se procházet nesmí. V atributové tabulce výsledné cesty jsou pak vidět akumulované hodnoty nákladového atributu, pokud je tento časový tak případně i čas přesunu.

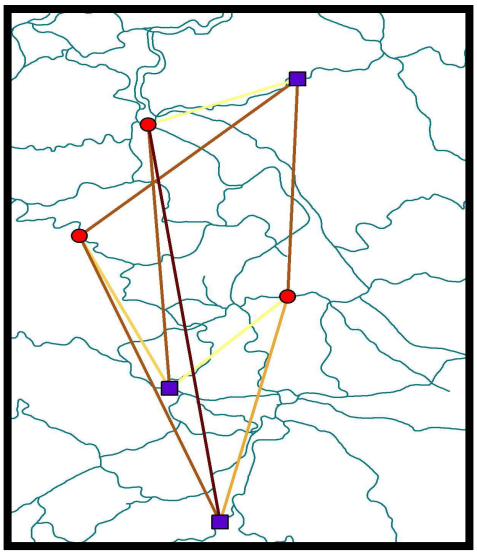
- **Closest Facility** – Tato analýza je podobná hledání cesty, ale na rozdíl od ní vytváří jedinečnou cestu od každého počátku k nejbližšímu konci – obslužnému zařízení. Počet těchto obslužných zařízení je nastavitelný, tj. se nemusí jednat jen o 1,ale např. o 3. Nastavitelný je též směr – buď od obslužné jednotky či k ní – toto má význam tehdy, když jsou definovány různé nákladové atributy pro směry. Jinak je nastavení obdobné jako u hledání cesty, ještě je zde navíc položka CutOff Value – tj. maximální vzdálenost, za kterou už program obslužné jednotky nehledá; Vytvořené cesty mají v atributu jak počáteční bod, tak číslo obslužného místa, proto pomocí symbology je lze např. barevně odlišit.

- **Service Area** – Pro daná místa generuje území, které „obslouží“ na základě nákladového atributu. Výsledek je prezentován buď liniemi či polygonem (viz. Obr. 5). Základním parametrem pro tuto analýzu je hodnota zlomu (break value), která vymezuje, jak velké území daná lokace obslouží. Opět zde jde nastavit směr pohybu, který má význam při různě definovaném nákladovém atributu pro oba směry. Pokud uživatel nastaví vykreslení polygonů může nastavit jestli každé zadané místo bude mít své vlastní polygony, či polygony budou společné pro všechna zadaná místa. Dále jsou polygony odstupňovány podle toho, jak jsou dosažitelné až po break value (bod zlomu). Lze nastavit, aby každý polygon reprezentující oblast s horší dostupností obsahoval i území s dostupností menší, a tedy polygon s maximální

dostupností obsahoval celé obslužné území (tj.model disku). Alternativně lze zvolit ring model, kdy polygony vytvářejí prstence. Podobným způsobem lze vizualizovat i linie. Zde se dá nastavit, zda-li se budou linie pro jednotlivé lokace překrývat či ne. Jinak je většina nastavení podobná těm, která jsou použitá i u předchozích analýz.



Obr. 5 Obslužná území - linie
(zdroj: ESRI ArcMap 9.1)



Obr. 6 Nákladová matice
(zdroj: ESRI ArcMap 9.1)

- **OD cost matrix** - Spíše schematický analytický nástroj, který dvojici počátek – konec, přiřazuje linii(lze přiřadit jen rovnou linii, proto schematický nástroj). Nejvyšší možný počet takovýchto linií je roven n=počet počátků * počet konců, ale lze nastavit i menší počet konců. Opět zde figuruje CutoffValue. Pro to aby tato metoda vizuálně poskytla informace je třeba upravit symbologii výsledných linií, a vizualizovat výsledky dle atributu akumulovaná cena (nebo jiný nákladový atribut) (viz. Obr. 6).

4.2 ILWIS

Možnosti tohoto softwaru jsou z hlediska analýz prostorové dostupnosti menší, než v případě ArcGISU. Kromě nástrojů pro tvorbu hydrogeologických výstupů (řešících plochu povodí, průběh vodních toků apod.), kterými se práce nezabývá se zde nalézá v podstatě jediný nástroj:

- **Distance Calculation -** Tento nástroj umožňuje provedení 2 funkcí + možnost rozšíření analýzy o Thiessenovy polygony. Jsou to funkce analogické k funkcím v prostředí ArcGIS – Spatial Analystu, a to Cost Distance a Euclidean Distance, v případě Thiessenových polygonů navíc ještě Cost Allocation a Euclidean Allocation. Možnosti nastavení analýzy jsou poněkud chudší a umožňují pouze stanovit do jakého atributu se bude výsledná vzdálenost (či obecně náklad) ukládat a v jakém rozmezí tohoto atributu má analýza probíhat (obvykle 0-x). Oproti ArcGISu lze stanovit přesnost, s jakou bude tento atribut počítán. Klíčovou vlastností je to, že

vstupem musí být raster, kdy všechny pixely jsou výchozí body, s výjimkou těch se záznamem NoData, které jsou brány jako cílové body. Tato vlastnost si vyžaduje většinou další krok při přípravě dat a to konverzi zpravidla bodové vrstvy na rastrovou.

4.3 GRASS

GRASS nabízí množství funkcí, které dovolují uživateli pracovat jak v rastrovém tak ve vektorovém modelu. Pro práci s rastry se zde nacházejí následující funkce:

- **Cost surface –** Tento nástroj je podobný Distance Calculation u ILWISu a Cost Distance u ArcGISu. Umožňuje nastavit nákladový povrch, výchozí body v rastrovém či vektorovém formátu a hodnotu, po jejímž dosažení již algoritmus dále nepočítá (tj. maximální náklady). Navíc, což u jiných nástrojů k dispozici není, zde lze nastavit tzv. Knight's move, tj. 16-směrný pohyb, který analýzu zpomaluje a zpřesňuje (viz. kapitola o algoritmech). Taktéž lze nastavit cílové body; po té co je dosaženo všech cílových bodů algoritmus končí. Nástroj umožňuje nastavit též způsob, jakým se bude pracovat s pamětí.

- **Cumulative movement cost –** Obdobou Path Distance u ArcGISu. Do analýzy přidává ještě DEM na jehož základě je počítána skutečná ušlá vzdálenost a do úvahy jsou brány i náklady při chůzi na různě skloněných površích.

Analogií Network Analystu u ArcGIS je několik funkcí u GRASSu. Narozdíl od ArcGISu není v GRASSu potřeba mít připravený Network Dataset. Stačí mít libovolnou vektorovou mapu, v níž jednu vrstvu tvoří linie (tedy hrany) a druhou uzly.

- **Networ maintenance –** Tento nástroj slouží k přípravě vektorové mapy pro analýzu. Kromě přípravy reportů o síti má uživatel dvě možnosti. Buď k existující síti nechá vygenerovat uzly (na koncích jednotlivých liniových objektů), nebo naopak přidá do sítě hrany, aby předem dané body zapojil do sítě. Tímto je zaručena použitelnost takto vzniklé vektorové mapy pro další analýzy.

- **Allocate subnets –** Vytváří síť, kde každé linii je přiřazeno číslo kategorie bodu, který toto místo „obsluhuje". Výsledkem je na první pohled identická síť s tou, která vstupovala do analýzy, liší se právě jen přiřazenou kategorií. Uživatel může zadat náklady v obou směrech, náklady pro uzly a povinně zadává uzly, které budou sloužit jako centra – obslužné jednotky.

- **Shortest route –** Vytváří liniovou vrstvu, která znázorňuje cestu s nejmenšími náklady od jendoho bodu k druhému (tyto musí být buď uzly, či mohou být zadány v souřadnicích a nacházet se v zadané toleranci). Uživatel může nastavit zda-li výsledná vrstva obsahuje jen jeden objekt – tedy nejkratší cestu, nebo objektů více, v souladu s tím jak byly rozděleny linie vektorové mapy, která vstoupila do analýzy. Stejně jako u minulé analýzy lze zadat náklady v obou směrech procházení hran a též náklady pro uzly.

- **Split net –** Podobná analýza jako subnets, akorát zde není liniím přiřazováno číslo kategorie bodu, ale číslo kategorie pásma vzdálenosti, do kterého linie spadá (tj. např. 0-5 km bude kategorie 1; 5-10 km kat. 2 apod. Stejně se dá použít i časových údajů.). Možnosti nastavení stejné jako u subnets, navíc je nutno nastavit zlomové body pro zařazení do kategorií.

- **Steiner Tree –** Vytvoření optimální sítě pro dané body. Používá se např. při pokládání kabelů apod. Výsledkem je síť, která je optimální co do nákladů (pokud nezadány, tak optimální co do délky – tj. nejkratší možná). Možnosti nastavení jsou obdobné jako u ostatních analýz.

- **Traveling Salesman Analysis –** Problém obchodního cestujícího. Nachází optimální trasu, která spojuje sérii zadaných bodů (na pořadí procházení nezáleží). Nastavení stejné jako u ostatních analýz. Výstup je podobný tomu ze Steinerova stromu, ale nemusí být při zadaných stejných bodech vždy stejný.

4.4 Srovnání funkcionality softwarových nástrojů

Je logické, že ze 3 hodnocených software nabízí nejvíce možností a funkcí ArcGIS, vzhledem k tomu, že se jedná o placený software. V některých případech však i volně dostupné programy nabízí nějakou funkci, kterou ArcGIS nenabízí. Na tyto možnosti bude zaměřena tato kapitola. Budou zde také tabulky uvádějící přehled nástrojů jednotlivých software, aby vynikly analogie mezi nástroji ArcGIS, ILWIS a GRASS.

V rastrovém formátu má ILWIS navíc možnost volby přesnosti, s jakou jsou počítána výstupní data, což umožňuje uživateli volit mezi přesností a rychlostí. GRASS v tomto ohledu nabízí možnost provádět analýzu se zaškrtlou možností Knight's move, která nabízí přesnější výsledky za delší čas. Výhodou je též možnost nastavit cílové body, po jejichž dosažení analýza dále neprobíhá. Nevýhodou je nemožnost provádět alokační funkce. Funkce shrnuje Tab. 3.

Tab. 3 Nástroje pro rastrová data – srovnání

ArcGIS	ILWIS	GRASS
Cost Distance	Distance Calculation	Cost Surface
Cost Allocation	Distance Calculation	-
Cost Backlink	-	-
Euclidean Distance	Distance Calculation	-
Euclidean Allocation	Distance Calculation	-
Euclidean Direction	-	-
Path Distance	-	Cumulative Movement Cost
Path Distance Allocation	-	-
Path Distance Backlink	-	-

Ve vektorovém formátu je hlavní předností GRASS jednoduchost, s jakou lze upravit liniový atribut pro použití v síti. Postup u ArcGISu je mnohem složitější. Dále nabízí GRASS navíc funkci na vytvoření optimální sítě. Nevýhodou je pak nemožnost zvolit si libovolný výchozí bod, ale pouze některý z uzlů a celkově méně možností při provádění analýzy. Srovnání funkcí ukazuje následující tabulka (Tab. 4).

Tab. 4 Nástroje pro vektorová data - srovnání

ArcGIS	**GRASS**
Route	Shortest route
Route	Traveling Salesman
Service Area	Allocate Subnets
Service Area	Split Net
Closest Facility	-
OD Cost Matrix	-
-	Steiner Tree

KAPITOLA 5
Algoritmy používané při analýzách

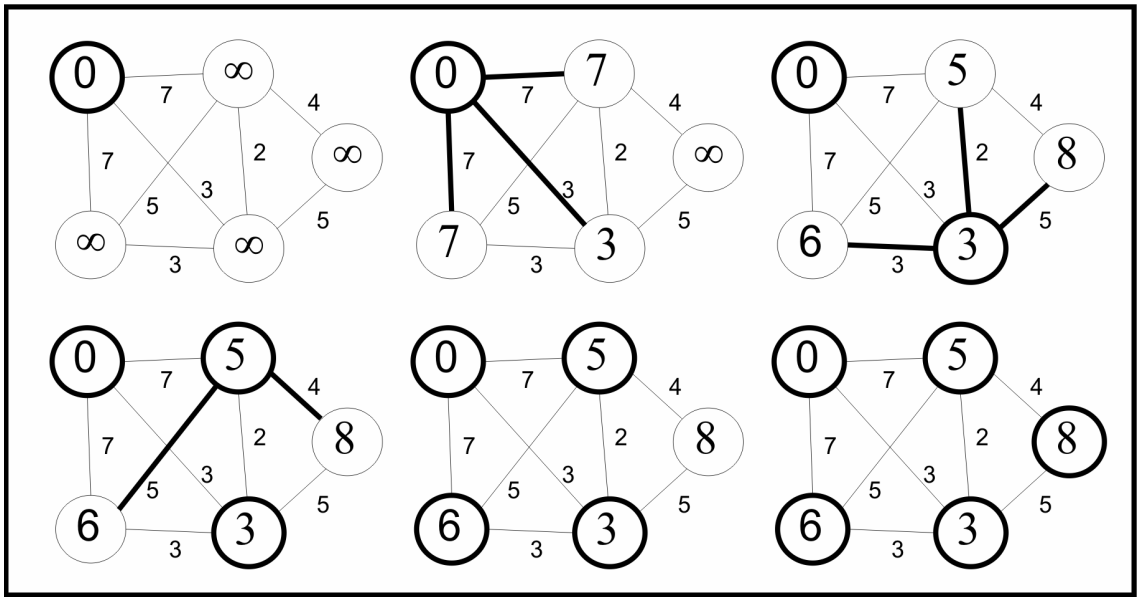
Tato kapitola se věnuje popisu několika základních algoritmů používaných při analýzách prostorových dostupností. Na první pohled se jeví přístupy k vektorovým a rastrovým datům jako naprosto rozdílné. Při bližším prozkoumání se však ukazuje, že lze na oba tyto datové modely aplikovat podobné metody.

5.1 Algoritmy pro práci s vektorovými vstupními daty

Při práci s vektorovými daty se uplatní algoritmy, které slouží k řešení síťových analýz, vycházejících z teorie grafů.

Prvně jsou to algoritmy, sloužící k nalezení nejkratší cesty, přičemž lze i ukládat kumulované náklady. Jsou to v současnosti asi nejužívanější Dijkstrův algoritmus, dále Bellman-Fordův algoritmus a Floyd-Warshallův algoritmus. Dále pak jsou to pak algoritmy určené pro nalezení minimální kostry grafy. Jedná se o Borůvkův algoritmus, Kruskalův algoritmus a Jarníkův algoritmus. Práce se těmito 3 algoritmy nebude dále zabývat, protože problém hledání minimální kostry grafu neodpovídá zadání práce.

- **Dijkstrův algoritmus** lze použít pro procházení grafu s nezáporně ohodnocenými hranami. Začíná od výchozího bodu, kterému přiřazuje akumulované náklady 0, všem ostatním uzlům přiřazuje zatím neznámou akumulovanou hodnotu(nekonečno). Všem sousedům výchozího bodu ukládá místo nekonečna hodnotu danou ohodnocením spojujících hran. Zároveň je zařazuje do listu. Uzel s minimální hodnotou je vyřazen z listu a zařazen do pole výsledků. Na list jsou zařazeny jeho sousedící uzly i s akumulovanou hodnotou; v případě že už zde jsou je jim aktualizována hodnota akumulovaných nákladů (pokud je nižší než původní, již uložená). Poté je z listu opět vyřazen uzel s minimální hodnotou a takto algoritmus pokračuje až do dosažení všech cílových bodů či všech uzlů v grafu (viz Obr. 7).



Obr. 7 Dijkstrův algoritmus

(Zdroj: vlastní (inspirace např. BAYER, 2008))

Na následujícím schématu je zachycen princip fungování algoritmu. Jedná se o obdobu podobných schémat prezentovaných při vysvětlení Dijkstrova algoritmu. Nejprve je hodnota výchozího bodu v levém horním rohu inicializována na 0, a tento bod přidán do skupiny uzavřených bodů(tučně okroužkované body). Poté jsou inicializovány všichni jeho sousedi na jeho hodnotu (tj. 0) zvětšenou o hodnotu hran, které k těmto sousedním bodům vedou. Posléze je opět vybrán bod s nejnižšími náklady a postup se opakuje. Ve třetím kroku vidíme, že dochází k úpravě již jednou inicializovaných hodnot, protože nové hodnoty jsou nižší. Algoritmus pokračuje dokud nejsou všechny body ve skupině uzavřených bodů.

- **Bellman-Fordův algoritmus** je pomalejší, ale lze ho použít i pro procházení grafem se záporně ohodnocenými hranami. Jeho průběh je rozdílný v tom, že uzel, který je v Dijkstrově algoritmu vyřazen z listu zde vyřazen není a může být opětovně navštíven s nižšími náklady. Průběh je takový, že se opakovaně (tolikrát, kolik je uzlů), prochází všechny uzly a aktualizuje se jejich hodnota akumulovaných nákladů.

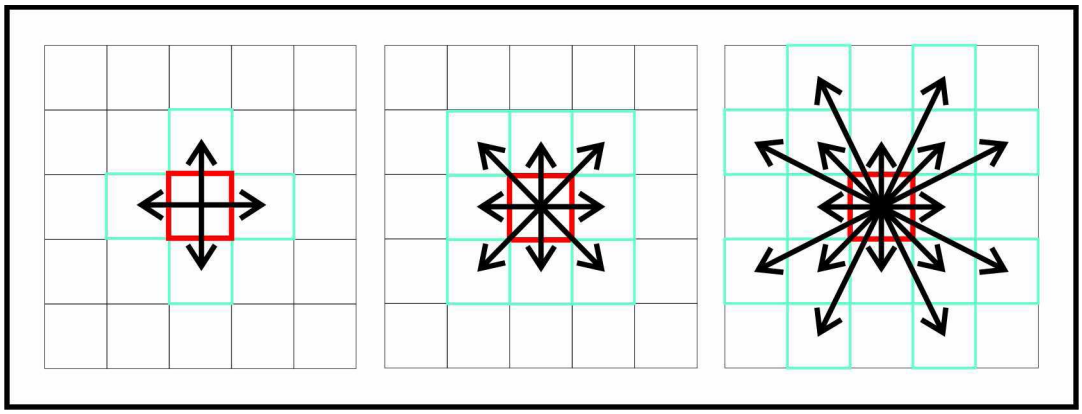
- **Floyd-Warshallův** algoritmus počítá nejkratší cestu mezi všemi dvojicemi uzlů grafu pomocí rekurze. Používá se u analýz , kde se počítá s velkým počtem jak výchozích tak cílových bodů. Výsledkem je pak část matice, která takto vzniká (popřípadě celá matice pokud zjišťujeme náklady z každého bodu do každého). (popisy algoritmů např. ČERNÝ, 2008)

5.2 Algoritmy pro práci s rastrovými vstupními daty

Algoritmy pro práci s rastrovými daty jsou založené na podobném principu. Každý pixel rastru lze chápat jako uzel, který spojuje 4-16 hran (vysvětleno dále, obvyklý počet je 8, tj. 8 směrů z pixelu – 4 přímé a 4 diagonální). Ohodnocení těchto hran není dáno atributem, ale odvozuje se z nákladového povrchu a vzdálenosti uzlů mezi sebou. Uzly sousedící v přímém směru jsou vzdálené 1*velikost pixelu, zatímco uzly sousedící v diagonálním směru jsou vzdálené √2*velikost pixelu. Vzhledem k tomu, že je možné takovéto připodobnění, je též možné použít stejných algoritmů jako v případě vektorových dat. Vzhledem k tomu, že počet uzlů je většinou větší než u vektorových dat (pro rastr 1000×1000 pixelů je to 1000000 uzlů), používá se prakticky pouze Dijkstrův algoritmus, který i v případě větších rastrů je pomalý, proto se přistupuje k využití heuristických metod (viz níže).

V předchozím odstavci bylo zmíněno 4-16 hran, tedy 4-16 pixelů se kterými může daný pixel sousedit (viz. Obr. 8). V případě, že sousedí jen se 4, jedná se o tzv. rook's move, který počítá jen se 4 přímými směry(nahoru, dolů, vlevo, vpravo). Akumulovaná hodnota nákladů pro sousedící buňku se spočte jako: Akumulovaná hodnota nákladů pro danou buňku pro danou buňku + (hodnota nákladů pro akt. buňku+hodnota nákladů pro sousední buňku)/2*.

V případě, že bereme v úvahu 8 směrů jedná se o tzv. Queen's move (směry jako u rook's move + vlevo-nahoru, vlevo-dolu, vpravo-nahoru, vpravo dolu). Akumulovaná hodnota nákladů pro sousedící buňku se spočte jako: Akumulovaná hodnota nákladů pro danou buňku pro danou buňku + (hodnota nákladů pro akt. buňku+hodnota nákladů pro sousední buňku) *√2 / 2.



Obr. 8 Možnosti pohybu v rámci rastru

Některé programy dovolují brát v úvahu 16 směrů. Jedná se o tzv. Knight's move (viz. diagram). Pojmenování získal díky tomu, že 8 ze 16 pohybů připomíná pohyb koně po šachovnici. Akumulovaná hodnota nákladů pro takto „sousedící" buňky (i když spolu dle běžného chápání nijak nesousedí) se vypočte vzorcem, který bere v úvahu délku spojnice (√5) a průměrnou hodnotu nákladového povrchu všech 4 dotčených buněk. Používá se pro zpřesnění analýzy, zejména při delších pohybech, které se odlišují od diagonálního či přímého směru.

- **Dijkstrův algoritmus** funguje pak stejným způsobem jako u vektorových dat. Jediný rozdíl je v tom, že sousední buňky (uzly) nejsou hledány na základě hran, ale na základě toho, jaký typ pohybu jsme zvolili, tudíž těchto sousedních bodů je buď 4,8 nebo 16.

Při vzniku tohoto rastru zároveň může vznikat i tzv. back-link rastr, či alokační rastr (viz. Popis softwarových nástrojů). Back-link rastr vzniká tím způsobem, že při každém zařazení buňky do výsledného rastru je zároveň i zapsán směr, kde se od této buňky nachází buňka s nejnižšími náklady, tj. buňka ze které algoritmus k dané buňce došel. Stejně tak je zapisována alokační hodnota buňky, ze které algoritmus k dané buňce došel. Pro výchozí body je tato hodnota nastavena na libovolné číslo reprezentující tento bod (např. jeho ID v databázi).

- **A* algoritmus**, je založen na Dijkstrově algoritmu, ale obsahuje i heuristickou složku, která odhaduje vzdálenost k cílovému bodu. Na základě toho, prochází jen body, u kterých je pravděpodobné že vedou k cíli. Je vhodný pouze pro problémy, kde jsou specifikované cílové body, a to v malém množství (ideálně 1). Kumulované náklady se počítají následovně: $C = G + F$, kde G jsou kumulované náklady bodu počítané stejně jako u Dijkstrova algoritmu a F jsou předpokládané náklady na dosažení cílového bodu z daného bodu, pro nějž se výpočet provádí. Způsobů, kterými se odhadují náklady na dosažení cílového bodu, je mnoho. Ne vždy může tento způsob přinášet přesné výsledky. (Celý tento odstavec LESTER, 2005)

Celkově lze konstatovat, že univerzálním algoritmem je Dijkstrův algoritmus. U analýz nad vektorovými daty, lze uvažovat při větším množství výchozích a cílových bodů o Floyd-Warshallově algoritmu. Pokud existují i záporně ohodnocené hrany je třeba zapojit Bellman-Fordův algoritmus. U rastrových dat lze v případě složitějších povrchů, kde se např. nalézají překážky apod. zvažovat nasazení algoritmu A*.

KAPITOLA 6
Návrh vlastního programu pro řešení analýz

Součástí práce je i program Analýzy prostorové dostupnosti, který byl naprogramován autorem práce. Program Analýzy prostorové dostupnosti, vznikl z těchto důvodů:

- srovnání různých algoritmů s různými parametry z hlediska časové náročnosti
- srovnání výstupních dat těchto algoritmů z hlediska přesnosti

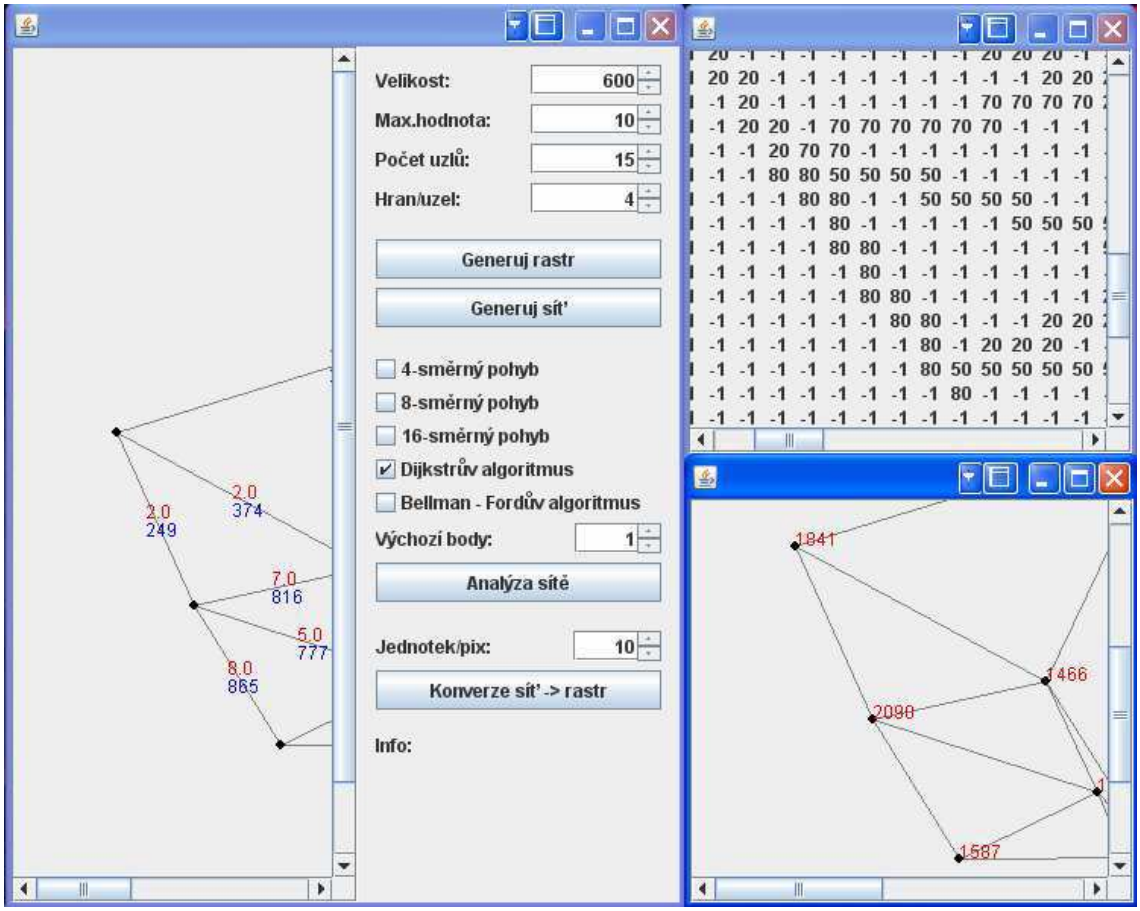
6.1 Funkce programu

Tato podkapitola by měla uživatelům programu poskytnout návod k jeho použití. Nejprve zmíním v několika bodech klíčové vlastnosti programu:

- vstupní data jsou generována vždy náhodně dle zadaných parametrů
- program umí generovat jak rastry, tak sítě
- program umí převádět sítě na rastry v nastavitelném kompresním poměru
- nad rastrovými daty provádí program Dijkstrův algoritmus se 3 různými parametry – bere v úvahu 4-směrný, 8-směrný a 16-směrný pohyb
- nad vektorovými daty provádí program Dijkstrův a Bellman-Fordův algoritmus
- počet výchozích bodů lze volit
- program zobrazuje časy provádění jednotlivých algoritmů

Před provedením analýz je nutno nejprve vygenerovat síť či rastr. Při generování rastru je použit pouze parametr velikost, který určuje rozměry rastru. Ten je vždy čtvercový a obsahuje tedy velikost · velikost pixelů. Parametr max.hodnota pak určuje, jaká je nejvyšší hodnota nákladů pro jeden pixel. Program toto číslo generuje náhodně. Číslo je celé a může nabývat jakékoliv hodnoty od 1 až po daný parametr. Velikost při tvorbě sítě určuje v jak velkém prostoru bude síť vybudována. Jediné co je ještě třeba osvětlit je parametr hran/uzel. Ten udává střední hodnotu hran, které bude mít jeden uzel. Jakmile je hrana ohodnocena, zobrazí se toto ohodnocení u hrany červeně, zatímco modře se zobrazují celkové náklady na projití hrany, které reflektují i její délku.

Poté již mohou být prováděny analýzy. Ty se otevírají v novém okně. Výchozí body se generují zcela náhodně. Konverze probíhá tak, že ze sítě o velikosti 600 vytvoří při kompresním poměru 10 (jednotek/pix) rastr 60×60 pixelů. Poté lze nad rastrem, který takto vznikl i nad sítí provádět všech pět analýz. Na Obr. 9 je vlevo základní panel s vytvořenou sítí, vpravo nahoře rastr vzniklý konverzí a vpravo dole výsledek za použití Dijkstrova algoritmu.



Obr. 9 Program „Analýzy prostorové dostupnosti“

6.2 Konstrukce tříd

Program je naprogramován v jazyce Java a sestává ze 7 tříd. Celý běh programu se odehrává v hlavním JFramu, jehož vzhled a chování určuje třída Hlavni. Pro provádění analýz nad vektorovými daty je zde třída Sit, která dále sestává z tříd Uzel a Hrana. Výsledky analýzy nad vektorovými daty jsou pak třídy VyslSit (díky značným odlišnostem od sítě vstupních dat). Pro analýzy nad rastrovými daty je zde třída Rastr, která dále sestává z třídy Pixel. Výsledky analýz nad rastrovými daty jsou též třídy Rastr, vzhledem k malým odlišnostem výstupních i vstupních dat. Následuje popis jednotlivých tříd s popisem jejich proměnných a nejdůležitějších metod.

Třída **Hlavni** obsahuje kromě komponent, generovaných vývojovým prostředím, následující proměnné:

- Rastr rastr – je zde uložen aktuální rastr, pokud zatím žádný vytvořený není, je ošetřeno, aby se s ním nedaly provádět žádné analýzy.
- Sit sit – je zde uložena aktuální vektorová síť. Platí to samé co u rastru.
- Uzel [] startovniUzly – jsou zde uloženy výchozí body (uzly)
- Pixel [] startovniPixely – jsou zde uloženy výchozí body (pixely)
- int pomer – pokud došlo ke konverzi vektorových dat na rastrová, je zde uložen převodní poměr.
- JFrame [] ramy – pole rámů, do kterých se ukládají výsledky analýz

Tato třída neobsahuje žádné významnější metody. Jejím jádrem jsou 4 metody, které reagují na stisk každého z tlačítek, tj. vytvoření sítě, vytvoření rastru, provedení analýz a konverze sítě. Dále je zde metoda void setComponents(), která JSPinnerům nastavuje přípustné hodnoty a metoda Pixel [] generujNahodneBody, která z již vygenerovaných výchozích uzlů tvoří pixely o stejných souřadnicích (pro srovnatelnost výstupních dat při analýzách jak nad rastrovými tak nad vektorovými daty).

Síť je tvořena z uzlů a hran. Vzhledem k tomu, že třída **Hrana** je méně rozsáhlá, bude nyní následovat její popis. Třída má následující proměnné:

- Uzel u1,u2 – 2 uzly, které jsou koncovými body hrany
- int atribut – hodnota přiřazená hraně, jednotkové náklady na její projití (celkové náklady je pak třeba ještě nutno vynásobit délkou hrany)
- int indexHrany – každá hrana má svůj index
- static int pocet – Počet existujících hran
- static Hrana[] hrany – všechny existující hrany

Významný na této třídě je pouze její konstruktor (předávají se 2 uzly a max.hodnota, kterou může atribut hrany nabýt). Je v něm generována hodnota atributu a zároveň přiřazena 2.uzlu tato hrana (prvnímu je přiřazena již v rámci volání konstruktoru). Následuje již jen sada metod na nastavování a získávání atributů a nulování počtu hran.

Složitější je třída **Uzel**. Obsahuje následující proměnné:

- int x,y – souřadnice uzlu
- int index – index uzlu
- int atribut – kumulované náklady pro uzel, použije se pro tvorbu objektu třídy VyslSit
- Hrana [] hrany – hrany navázané na uzel
- int pocetHran – skutečný počet hran

- int maxPocetHran – maximální počet hran, uplatňuje se při vytváření sítě
- boolean pridano – určuje zda-li je už uzel součástí výsledné sítě
- static int pocetUzlu – počet uzlů

Důležité metody:

- boolean isConnected (Uzel u) – vrací true, pokud je uzel spojený hranou s uzlem u
- Uzel [] najdiSousedy() – danému uzlu najde sousední uzly, které dosud nebyly přidané do výsledné sítě

Dále již třída obsahuje jen settery a gettery, plus méně významné pomocné nástroje. Nejrozsáhlejší ze všech je třída **Sit**, obsahuje následující proměnné:

- Uzel [] uzly – Základní kostra sítě, všechny uzly, které síť obsahuje
- int souradMax – maximální hodnota souřadnic
- int hranyStred – průměrný počet hran, které má jeden uzel
- int pocetUzlu – počet uzlů
- int max – maximální hodnota, které bude nabývat atribut hran
- ArrayList<Uzel> fronta – dynamická datová struktura používaná při běhu Dijkstrova a Bellman-Fordova algoritmu

Již v konstruktoru jsou vygenerovány uzly a následně k nim i hrany. Třída obsahuje následující významnější metody:

- paintComponent – přetížená metoda třídy jComponent, slouží k vykreslení sítě na obrazovku
- void generujUzly(),void generujHrany() – metody použité v konstruktoru při vytváření sítě
- Uzel [] najdiNejblizsi(int pocet,Uzel u) – k uzlu u hledá nejbližší uzly, jejich počet je jedním z argumentů metody.
- Double [][] getVzdalenosti(Uzel u) – počítá vzdálenosti od uzlu ke všem uzlům, výsledkem je dvojrozměrné pole, kde první rozměr tvoří vzdálenost a druhý index příslušného uzlu, ke kterému je počítána
- VyslSit dijkstra (Uzel [] startovni) – metoda pro získání výstupních dat ze vstupních. Nejprve jsou inicializovány startovní uzly (viz. metoda inicializaceVychozich), a posléze již je jen vybírán první prvek z fronty a zařazován do výsledné sítě(viz. metoda pridejDoVysledne). Přitom je měřen čas, který je pak výsledné síti předán.
- void inicializaceVychozich(Uzel [] startovni, VyslSit vysledek) – všem startovním bodům je nastavena kumulovaná hodnota (atribut) na 0 a všichni jejich sousedi jsou přidáni do fronty.

- void pridejDoVysledne (Uzel u, VyslSit vysledek) – uzel u je přidán do výsledné sítě, všichni jeho sousedi jsou přidáni do fronty (resp. aplikuje se na nich metoda pridejDoFronty, což ještě nutně nemusí znamenat přidání do fronty). Zároveň je uzel u vyřazen z fronty

- void pridejDoFronty(Uzel vychozi,Uzel pridavany) – pokud přidávaný uzel ještě není ve frontě je tam přidán s atributem výchozího uzlu + atributem jejich spojující hrany, v případě že se uzel ve frontě již nachází je provedeno ověření, zda-li je jeho už uložený atribut nižší než atribut, který by vznikl sečtením atributu výchozího uzlu s atributem hrany. Pokud ne, je atribut přidávaného uzlu upraven.

- Rastr konverze (int pomer) – převádí síť na rastr, původní velikost sítě je zmenšena dle hodnoty pomer, tzn. rastr je menší než síť.

- VyslSit bellman (Uzel [] startovni) – vytváří výslednou síť mnohonásobným procházením sítě

- Uzel [] generujNahodneUzly (int pocet) – generuje náhodné uzly, které pak slouží jako výchozí body.

Třída **VyslSit** obsahuje pouze proměnné double cas a Uzel [] uzly, do které se ukládají uzly, které jsou již vyřazeny z fronty a zařazeny do výsledné sítě. Nejvýznamnější je u této třídy metoda paintComponent, která zajišťuje vykreslení výsledné sítě.

Třída **Pixel** slouží k vybudování třídy Rastr. Obsahuje následující proměnné:

- int x,y – souřadnice pixelu
- int hodnota – náklady na procházení tímto pixelem
- int kumul – kumulované náklady – uplatní se v rastru přechovávajícím výsledky analýzy
- boolean pridano – indikuje, zda-li byl pixel již přidán do výsledného rastru

Celá třída sestává pouze ze setterů a getterů. Třída **Rastr** má následující proměnné:

- Pixel [] bunky – uchovává informace o jednotlivých pixelech
- int velikost – velikost mřížky, celkový počet pixelů je pak velikost*velikost
- int max – maximální hodnota, které mohou náklady na průchod jedním pixelem nabýt.
- ArrayList<Pixel > fronta – obdoba fronty u třídy Rastr, akorát místo uzlů zde figurují pixely
- double cas – čas za který proběhla analýza nad rastrovými daty

Důležité metody:

- void generujRastr() – zaplnění pole buněk hodnotami

- JPanel kresliBunky(boolean kumul) – vzhledem k tomu, že metoda kreslení přes paintComponent se zde ukázala jako nevhodná, tak se rastry kreslí přes tuto metodu. Výsledný

panel je pak přidán na komponenty vyšší úrovně. Argument kumul určuje zda-li se kreslí vstunpí rastr (pak argument nabývá hodnoty false) nebo výstupní rastr.

- Rastr dijkstra (Pixel [] startovni, int pohyb) – obdoba metody třídy Sit. Jediný rozdíl je zde v argumetnu pohyb, který určuje způsob, jakým se analyzují sousední pixely.

- inicializaceVychozich,pridejDoVysledne a pridejDoFronty jsou obdobou stejně pojmenovaných metod u třídy Sit.

- Pixel [] najdiSousedy (Pixel p, int pohyb) – hledá sousední pixely. Jejich počet určuje argument pohyb. Při hodnotě 0 má pixel 4 sousedy, při hodnotě 1 8 sousedů a při hodnotě 2 16 sousedů. Metoda z nich však vrací jen ty, které dosud nebyly přidané.

- void konvertovaneRastry (Rastr vysledek, int pohyb) – zaručuje převedení nepřístupných pixelů, vzniklých konverzí vektorů i do výsledného rastru.

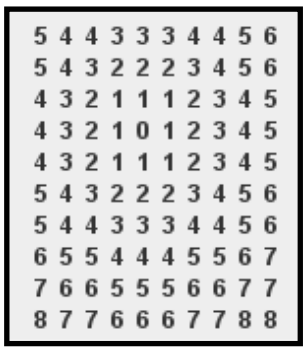
Program je funkční s několika omezeními. První z nich je velikost rastru, která je pro zobrazení maximálně 130×130 pixelů, pro plynulý chod je třeba rastr menší jak 500×500. V profesionálních GIS jsou samozřejmě analýzy nad takovýmito rastry vyřešeny během okamžiku. Omezujícím faktorem se zde jeví dynamické datové pole. To je však použito všude stejné, takže hlavní cíl, tedy srovnatelnost výsledků, program naplňuje.

KAPITOLA 7
Hodnocení algoritmů

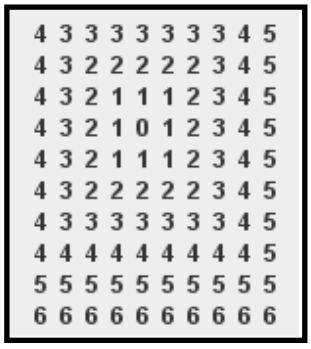
Tato kapitola využije program „Analýzy prostorové dostupnosti“ a pokusí se zhodnotit jednotlivé algoritmy při použití různých parametrů v následujících hlediscích: přesnost výstupních dat a časová náročnost.

7.1 Přesnost výstupních dat s ohledem na zvolený číselný formát

Program, který měl zhodnotit některé metody analýz pracoval s hranami, či pixely ohodnocenými kladnými celými čísly. Další výpočty však již byly prováděny s čísly reálnými. Pro potřeby vizualizace však byla zobrazována čísla v celočíselném tvaru. Při takto definovaném průběhu analýzy k žádným nepřesnostem, způsobeným zaokrouhlováním nedochází (samozřejmě kromě odchylky při závěrečné vizualizaci, jejíž relativní velikost závisí na rozptylu hodnot (pokud se kumulované náklady pohybují v řádu tisíců, nepředstavuje toto problém)). Pokud bychom všechny hodnoty kumulovaných nákladů brali jako celočíselné hodnoty, vzniklé zaokrouhlením, vznikala by ve výsledcích odchylka závislá jednak na počtu hran (či v případě rastrových dat pixelů), které je třeba překonat k danému bodu, pro který počítáme kumulované náklady, a také na tom, jakými hodnotami jsou hrany ohodnocené (čím větších hodnot budou nabývat, tím je relativní odchylka menší). Výsledkem by tedy bylo to, že u bodů, jež jsou vzdálenější od zdrojů, by vznikala větší odchylka.



Obr. 10: Kumulované náklady – double



Obr. 11: Kumulované náklady – integer

Na obrázcích 10 a 11 jsou znázorněny extrémní případy – na Obr. 10 probíhají operace nad datovým typem double, jen na závěr dochází k zaokrouhlení. Výsledky jsou tedy přesné. Na

Obr. 11 probíhají operace nad datovým typem integer, tj. vždy dochází k zaokrouhlení dolů. Zvolen byl nákladový povrch složený ze samých jedniček, tedy indiferentní. Vidíme, že na Obr. 11 dochází k nepřesnostem – odhadované kumulované vzdálenosti jsou nižší než by měly být. Výhodou je ovšem menší výpočetní náročnost a paměťová náročnost. Kompromisem mezi těmito dvěma přístupy je přístup, který sice všechna data uchovává v datovém typu integer, ovšem dochází zde vždy k zaokrouhlení. Takový přístup je stejně paměťově náročný jako v případě Obr. 11, ale je výpočetně náročnější. Ovšem vzhledem k tomu, že zaokrouhlování mění velikost dat oběma směry, v některých případech dochází k vyrušení efektu zaokrouhlení a tedy k přesnějším výsledkům. Ovšem konkrétně v tomto případě indiferentního povrchu tomu tak není, a je jedno jestli zaokrouhlujeme podle matematických pravidel, či vždy dolů.

U vektorových dat také lze uplatnit tyto 3 přístupy. Většinou však není důvod provádět analýzy nad datovým typem double. To by bylo opodstatněné pouze ve chvíli, kdy souřadnice budou nabývat nízkých maximálních hodnot, a taktéž hrany budou ohodnoceny nízkými hodnotami (tedy, hodnota atributu, který se postupně kumuluje bude nízká).

Celkově lze říci, že: Provádět analýzy nad datovým typem double je opodstatněné jen u nákladových povrchů, jejichž buňky nabývají nízkých hodnot a kdy nám záleží na přesnosti. V jiných případech stačí provádět analýzy nad datovým typem integer, a zaokrouhlovat – toto bude vhodným postupem v drtivé většině případů. Postup, kdy se nezaokrouhluje lze zvolit ve chvíli kdy nám záleží na rychlosti. Tento postup není vhodný pro nákladové povrchy, jejichž pixely nabývají nízkých hodnot.

7.2 Přesnost výstupních dat s ohledem na použitý algoritmus

U rastrových dat byl testován pouze jeden algoritmus, zato ale s rozdílnými přístupy k sousedním buňkám. Použití každého z těchto přístupů přináší poněkud odlišné výsledky. U vektorových dat byly použité algoritmy dva, ale ve výsledcích se neliší. Jeden je konstruován pro práci se záporně ohodnocenými hranami. V případě kladně ohodnocených hran však produkuje stejné výsledky jako první, akorát v pomalejším čase. Tato kapitole tedy srovná 3 různé přístupy k sousedním buňkám u Dijkstrova algoritmu nad rastrovými daty. Nejprve bylo provedeno srovnání nad indiferentním nákladovým povrchem. Na všech obrázcích je znázorněn výřez výsledného rastru, a to konkrétně pravý horní roh. Výchozí bod analýzy se od tohoto čtverce nachází vlevo dole, téměř v levém dolním rohu. Rastr má velikost 30×30 pixelů. Obr. 12 ukazuje výsledky pro 4-směrný pohyb. Je vidět, že výsledné hodnoty jsou vyšší než u ostatních druhů pohybu. 8-směrný (Obr. 13) a 16-směrný (Obr. 14) pohyb se pak liší již jen nepatrně. U 8-směrného jsou hodnoty nepatrně vyšší a to zejména v pravém dolním a levém horním rohu. Je to dáno tím, že tyto oblasti se nacházejí v přibližně 30-ti a 60-ti stupňovém úhlu od výchozího bodu. V těchto oblastech je analýza provedená pomocí 8-směrného pohybu nejméně přesná a naopak analýza provedená pomocí 16-směrného pohybu přesná. Ve 45-ti stupňovém úhlu

(pravý horní roh) se výsledky neliší a jsou přesné. Toto není dáno náhodou a platí to obecně pro všechny typy nákladových povrchů, zejména tam, kde hodnoty nákladového povrchu nemají příliš velký rozptyl.

37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37

Obr. 12: 4-směrný pohyb po rastru

31	32	32	32	33	33	34	34	34	35	35	36	36	37	37	37	38	38
30	31	31	31	32	32	33	33	33	34	34	35	35	36	36	36	37	38
29	30	30	30	31	31	32	32	32	33	33	34	34	35	35	35	36	37
28	29	29	29	30	30	31	31	31	32	32	33	33	34	34	35	36	37
27	28	28	28	29	29	30	30	30	31	31	32	32	33	34	35	36	37
26	27	27	27	28	28	29	29	29	30	30	31	31	32	33	34	35	36
25	26	26	26	27	27	28	28	28	29	29	30	31	32	33	34	35	36
24	25	25	25	26	26	27	27	27	28	28	29	30	31	32	33	34	35
23	24	24	24	25	25	26	26	26	27	28	29	30	31	32	33	34	35
22	23	23	23	24	24	25	25	25	26	27	28	29	30	31	32	33	34
21	22	22	22	23	23	24	24	25	26	27	28	29	30	31	32	33	34
20	21	21	21	22	22	23	24	25	26	27	28	29	30	31	32	33	34
19	20	20	20	21	21	22	23	24	25	26	27	28	29	30	31	32	33
18	19	19	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
17	18	18	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
16	17	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Obr. 13: 8-směrný pohyb po rastru

29	30	30	30	30	31	32	32	33	33	34	35	35	36	36	37	38	38
28	29	29	29	30	30	31	31	32	33	33	34	34	35	36	36	37	38
27	28	28	28	29	29	30	31	31	32	32	33	34	34	35	35	36	37
26	27	27	27	28	29	29	30	30	31	32	32	33	33	34	35	36	36
25	26	26	27	27	28	28	29	30	30	31	31	32	33	33	34	35	36
24	25	25	26	26	27	28	28	29	29	30	31	31	32	33	34	34	35
23	24	24	25	26	26	27	27	28	29	29	30	31	31	32	33	34	35
22	23	24	24	25	25	26	27	27	28	28	29	30	31	32	32	33	34
22	22	23	23	24	25	25	26	26	27	28	29	29	30	31	32	33	33
21	21	22	22	23	24	24	25	25	26	27	28	29	30	30	31	32	33
20	20	21	22	22	23	23	24	25	26	27	27	28	29	30	31	31	32
19	20	20	21	21	22	23	23	24	25	26	27	28	28	29	30	31	32
18	19	19	20	21	21	22	23	24	25	25	26	27	28	29	29	30	31
17	18	19	19	20	21	21	22	23	24	25	26	26	27	28	29	30	30
17	17	18	18	19	20	21	22	22	23	24	25	26	27	27	28	29	30
16	16	17	18	19	19	20	21	22	23	24	24	25	26	27	28	29	30
15	16	16	17	18	19	20	20	21	22	23	24	25	26	27	28	29	30
14	15	16	17	17	18	19	20	21	22	22	23	24	25	26	27	28	29

Obr. 14: 16-směrný pohyb po rastru

Další analýzy byly poté prováděny na náhodně vygenerovaném nákladovém povrchu, kde pixely nabývaly různých hodnot.

Výchozí body mají Obr. 15 a 16 v pravém horním rohu, opět při velikosti celého rastru 30×30. Je vidět že odchylky hodnot při použití jednotlivých způsobů se pohybují okolo čísla 10 (nejvíce 13). Není zde již patrný vliv úhlu od výchozího bodu. Stejných odchylek bylo dosaženo i při několikerém opakování se stejnou maximální hodnotou a jinými nákladovými povrchy.

Testování s jinými maximálními hodnotami přineslo podobné výsledky. Pokud na rastru 30×30 byl výchozí bod v levém horním rohu, v levém dolním rohu se hodnoty lišily přibližně o max.hodnotu nákladového povrchu/10. Při testování s max. hodnotou 100000 se již max. odchylka pohybovala okolo 3000. Dá se tedy předpokládat že čím vyšší max. hodnota a rozptyl tím menší relativní odchylka při porovnávání obou způsobů.

890	840	792	774	717	685	673	653	669	612	614	571	590	565
923	858	830	765	750	736	691	684	641	673	611	610	619	558
909	869	808	827	770	746	760	713	685	650	639	660	598	558
854	855	822	804	775	788	808	748	694	707	658	678	641	599
844	828	851	815	789	784	798	759	708	688	694	693	690	648
859	872	851	827	834	807	821	768	712	746	724	710	726	713
921	923	880	853	823	814	801	751	720	745	738	738	733	744
978	930	898	844	837	818	771	738	734	780	765	763	766	765
1003	976	897	835	847	795	753	779	762	810	774	773	785	764
1026	956	888	831	788	767	778	824	779	796	821	823	807	779
982	956	882	830	837	817	787	840	830	833	874	902	866	795
962	935	921	897	925	842	821	825	879	884	906	891	860	834
974	942	951	991	957	885	856	873	869	898	876	858	861	810
981	946	959	996	983	909	875	934	882	906	906	853	828	850

Obr. 15 Kumul. náklady – 8-směrný pohyb

885	833	785	767	709	678	666	646	661	604	607	564	582	557
911	851	823	757	743	729	683	677	634	666	603	603	611	550
901	861	800	820	763	739	752	705	677	643	631	653	591	551
846	847	814	797	768	780	801	741	687	698	650	671	633	592
836	821	843	808	782	777	790	752	701	680	686	685	683	640
851	864	843	820	826	799	814	761	704	739	716	702	719	698
914	916	873	845	815	806	793	743	713	738	730	731	725	735
970	922	891	837	829	811	763	730	727	772	758	755	758	756
996	969	890	827	839	787	746	771	754	803	766	765	778	755
1019	948	880	823	780	760	770	817	772	788	814	816	800	770
974	949	875	822	829	810	780	832	823	825	867	894	857	786
954	928	913	890	915	834	813	817	872	876	893	878	847	825
967	934	943	984	950	877	848	865	862	890	863	845	848	797
974	939	952	987	968	901	867	926	874	898	893	840	815	837

Obr. 16 Kumul. náklady – 16-směrný pohyb

Zajímavé je, že při testování na větších površích a tedy na větších vzdálenostech od výchozího bodu již k dalšímu zvětšování rozdílů nedocházelo(resp. docházelo ale v porovnání rastrů 30×30 a 60×60 jen ve velmi nepatrné míře). Ovšem kdybychom zvětšovali rastr na indiferentním nákladovém povrchu, zvětšovali by se odchylky lineárně.

Testování lze shrnout následujícím způsobem: 4-směrný pohyb se pro provádění analýz nehodí, protože odchylky jsou příliš velké, a nedá se jim předcházet. Použití 16-směrného pohybu je doporučeno tam, kde potřebujeme přesnější data a nezáleží na čase. V případě, že se jedná o hodně homogenní nákladový povrch, případně povrch, který nabývá málo hodnot je taktéž použití analýzy s 16-směrným pohybem odůvodněné.

7.3 Přesnost výstupních dat – rozdíl mezi analýzami nad rastrovými a vektorovými daty

Důvody proč převádět vektorová data na rastrová jsou uvedeny v předchozích kapitolách. Tato pasáž se pokusí srovnat analýzy prováděné nad rastrovými daty s těmi, prováděnými nad daty vektorovými. Aby byla zajištěna srovnatelnost je nutno nejprve vektorová data rasterizovat. Při tom taktéž dochází (většinou) ke zmenšení velikosti. Tj. vektorová data v prostoru 1000×1000 bodů, jsou např. konvertována na rastr 100×100 pixelů (při konverzním poměru 10). Úkolem této pasáže je mj. pokusit se objasnit, jaký má tento kompresní poměr a další faktory vliv na rozdílnost výsledků. Pro srovnání bude použit Dijkstrův algoritmus, v případě rastru s 8-směrným pohybem. Je tomu tak proto, že použití 16-směrného pohybu se v tomto případě jeví problematické z důvodu toho, že často nastává situace, kdy se při tomto pohybu prochází „neprůchodným“ pixelem. Dále se předpokládá, že analýza nad vektorovými daty bude vždy přesnější. Je to dáno tím, že u analýz nad rastrovými daty dochází k nepřesnostem v důsledku nepřímého pohybu, a také v důsledku možného „přeskakování“ z linie na linii. To nastává v případě, kdy blízko sebe vedou linie, které se nekříží, ale v rastrové analýze dojde k tomu, že algoritmus je bere jako sousedící. Pak se do celé analýzy zanáší dosti podstatná nepřesnost.

Nejvíce ovlivňuje přesnost výsledných dat to, jaká je „hustota“ uzlů na jeden pixel konvertovaného rastru. Je jasné, že pokud bude konvertována síť s 30 uzly na rastr 20×20

analýza díky velkým nepřesnostem v podstatě nebude možná. Další nutnou podmínkou je,aby nedocházelo ke křížení sítě mimo uzly. To znamená, že tento postup vylučuje mimoúrovňová křížení.

Na Obr. 17 je vidět výsledek analýzy nad konvertovaným rastrem, kdy byly splněny všechny výše jmenované podmínky, aby tato analýza měla vůbec smysl. Hodnota udávaná analýzou nad vektorovými daty udává v bodě u dolního okraje hodnotu 3149. Jde tedy o rozdíl okolo 10%. Jednalo se o analýzu, kde figurovalo pouze 6 bodů a výsledný rastr měl velikost 80×80 pixelů. V jiných částech rastru byla odchylka nižší.Při dalších analýzách a zachování stejných podmínek se výsledky značně lišily. V některých místech byly výsledky téměř bez odchylky, jinde byla značná. Pokud jsou rasterizované linie příliš blízko u sebe, dochází ke snižování hodnot akumulovaných nákladů. Proti tomu jde však jejich zvyšování v důsledku nepřímého pohybu. Na testovaných příkladech byly odchylky tak velké, že to vylučovalo praktické použití.

-1	-1	-1	-1	-1	4059	-1
-1	-1	-1	-1	-1	4009	-1
-1	-1	-1	-1	-1	3959	-1
-1	-1	-1	-1	-1	3909	-1
-1	-1	-1	-1	3839	3859	-1
-1	-1	-1	-1	3789	-1	-1
-1	-1	-1	-1	3739	-1	-1
-1	-1	-1	-1	3689	-1	-1
-1	-1	-1	-1	3639	-1	-1
-1	-1	-1	-1	3589	-1	-1
-1	-1	-1	3518	3539	-1	-1
3367	-1	-1	3468	-1	-1	-1
3377	3402	3422	3437	-1	-1	-1
-1	3412	3437	3450	-1	-1	-1
-1	-1	-1	3473	-1	-1	-1

Obr. 17: Kumulované náklady - konverze

Závěrem lze konstatovat, že použití rasterizace je z hlediska přesnosti dat značně diskutabilní. Tato metoda by mohla být využitelná při dostatečně malé hustotě linií, a při použití bufferu, pak by byl použitelný i 16 směrný pohyb, který by analýzu zpřesnil. Pokud k tomu není nějaký zvláštní důvod, není rasterizace linií vhodný postup, jak provádět analýzu.

7.4 Časová náročnost analýz

Jedním z cílů práce je i srovnat algoritmy na základě jejich časové náročnosti na provádění. Čas byl měřen zvlášť u analýz nad rastrovými a vektorovými daty z důvodu, který byl zmíněn v minulé kapitole – analýzy na rastrech, které vznikají konverzí z vektorových dat (liniových) nejsou ve většině případů použitelné, proto nemá smysl ani srovnávat jejich časovou náročnost. Měření bylo prováděno vždy na 5 různě vygenerovaných površích se stejnými parametry. V rámci každého povrchu pak byla provede 3 různá měření se 3 různými výchozími body. Výsledky těchto měření se nacházejí v přílohách, zde uvádím vždy jen průměrné hodnoty za všech 15 měření.

Tab. 5 Časová náročnost algoritmů

Nákladový povrch	Doba analýzy (v ms)	Nákladový povrch	Doba analýzy (v ms)
50 uzlů, 1 VB	0,138	20×20 pix,4 směry	0,585
50 uzlů,1 VB, Bellman	1,893	20×20 pix, 8 směrů	1,102
50 uzlů, 5 VB	0,187	20×20 pix, 16 směrů	2,436
500 uzlů, 1 VB	1,398	20×20 pix ,8 směrů, 10 VB	1,761
500 uzlů, 1 VB, Bellman	162,803	200×200 pix, 4 směry	387,739
500 uzlů, 1 VB, 20 hran	27,825	200×200 pix, 8 směrů	920,353
500 uzlů, 50 VB	3,264	200×200 pix, 16 směrů	2370,620
500 uzlů, 50 VB, Bellman	163,447		
5000 uzlů, 1 VB	23,964		
5000 uzlů, 1 VB, Bellman	6736,933		

Poznámka: VB = výchozí body, Bellman = Bellman-Fordův algoritmus, jinak Dijkstrův

Z dat (Tab. 5) je patrné, že s narůstajícím počtem uzlů stoupá časová náročnost u Dijkstrova algoritmu nad vektorovými daty přibližně lineárně, zatímco u Bellmanova algoritmu roste rychleji než lineárně. Ve všech případech je také tento algoritmus pomalejší, což se dalo předpokládat. Z dat vyplývá, že u Dijkstrova algoritmu s rostoucím počtem výchozích bodů roste i časová náročnost, ovšem pomaleji než lineárně. Poslední co lze konstatovat je, že když na každý uzel navážeme více hran analýza se logicky zpomalí.

Data z analýz nad rastrovými analýzami ukazují, že 8-směrný pohyb je na čas 2× náročnější než 4-směrný a 16-směrný 2× náročnější než 8-směrný. U větších rastrů se tento relativní rozdíl dále prohlubuje, tj. 16-směrný pohyb je oproti 8-směrnému náročnější víc jak dvakrát. S rostoucí velikostí rastru roste časová náročnost lineárně. S rostoucím počtem výchozích bodů náročnost taktéž stoupá.

Publikované časy je nutno brát pouze orientačně, pro porovnání s ostatními. Profesionální nástroje jako např. ArcGIS dosahují řádově rychlejších časů.

KAPITOLA 8
Závěr

Práce měla několik cílů. Prvním z nich bylo zmapovat dostupné nástroje pro tvorbu analýz prostorové dostupnosti. V této oblasti se povedlo kompletně zmapovat možnosti ArcGIS a jeho extenzí Spatial Analyst a Network Analyst. Práce by tedy měla přispět k tomu, že čtenáři usnadní jejich používání. Zároveň byly zmíněny alternativy v podobě systémů ILWIS a GRASS. Autor je toho názoru, že tyto 2 nástroje patří k nejvýznamnějším nástrojům z těch, které jsou dostupné volně. Lze konstatovat, že pro analýzu nad vektorovými daty je vhodný kromě ArcGISu pouze GRASS, který je snadněji použitelný než ArcGIS, a i jeho funkce jsou s ArcGISem srovnatelné. U analýz nad rastrovými daty pak ILWIS nabízí alternativu pro základní analýzy. Opět mohu vyzdvihnout jeho jednoduchost, ovšem v těchto typech analýz je jednoduchý na používání i ArcGIS. GRASS je v základních ohledech pro tyto analýzy použitelný také. Dokonce zvládá i složitější úlohy, ve kterých se bere v úvahu terén (DEM), po kterém pohyb probíhá. Rezervy má práce v analýze komerčních systémů, zejména MapInfa. Specializované nástroje (tedy ty, jenž nejsou komplexními GIS), byly autorem zkoušeny, ale pro potřeby práce se nehodily, proto, že buď nijak neobohacovali možnosti již zmíněné, či jejich možnosti byly vzdálené autorově definici analýz prostorové dostupnosti.

V další části jsou rozebrané algoritmy, pomocí kterých lze analýzy provádět. Je tomu tak proto, aby autor lépe mohl objasnit funkčnost svého programu. Jejich znalost je také důležitá při porovnávání různých algoritmů s různými parametry v závěrečné části práce. Jak se ukázalo, nejdůležitějším algoritmem pro provádění analýz je Dijkstrův algoritmus. Lze ho aplikovat jak na vektorová, tak na rastrová data. Popis je pak již jen stručně doplněn dalšími algoritmy, které však do velké míry mají s tímto algoritmem mnoho společného. Práce se samozřejmě nezabývala všemi algoritmy, pomocí nichž lze analýzy provádět. Některé byly jen zmíněny a o jiných autor usoudil, že nezapadají do rámce práce. Kromě Dijkstrova algoritmu byl zmíněn A* algoritmus pro práci se složitějšími daty, kde je známý výchozí i cílový bod. Pro práci s vektorovými daty pak Floyd-Warshallův algoritmus pro práci s velkým počtem výchozích a cílových bodů a Bellman-Fordův algoritmus pro práci se záporně ohodnocenými hranami.

Následuje popis autorova programu. Tento program splnil očekávání v tom, že lze snadno porovnávat časovou náročnost analýz. Co se týká hodnocení přesnosti, zde už je potřeba, aby

uživatel vynaložil jistou námahu, aby mohl přesnost u různých analýz srovnávat. Za autorovými očekáváními zaostal program v rychlosti (ve srovnání s profesionálními nástroji jako např. ArcGIS), z čehož plyne i to, že se nehodí pro větší analýzy (např. rastr 1000×1000 pixelů), které chtěl autor provést. Celkově však plní účel pro který byl napsán.

V závěrečné kapitole byly algoritmy s různými parametry hodnoceny z několika hledisek. Jako datový typ vhodný pro ukládání kumulovaných nákladů se ukázal integer, s tím že je nutno zaokrouhlovat. Pokud není zapotřebí natolik přesných dat, lze převod nechat na počítači (tj. zaokrouhlení dolů). Z hlediska přesnosti dat není rozdíl mezi Bellman-Fordovým a Dijkstrovým algoritmem. U Dijkstrova algoritmu nad rastrovými daty se ukázal nevhodný postup, který přiřazuje každému pixelu pouze 4 sousední. V drtivé většině případů si uživatel vystačí s 8-směrným pohybem. Konverze vektorů - linií na rastr se ukázala vhodná jen v omezeném množství případů. Z hlediska časové náročnosti u všech typů analýz stoupala náročnost při zvyšujícím se počtu výchozích bodů. Analýza se zapojením 16-směrného pohybu byla více jak 2× pomalejší než analýze se zapojením 8-směrného pohybu. Bellman-Fordův algoritmus se ukázal jako nevhodný pro větší analýzy (analýzy s větším počtem uzlů).

Domnívám se , že přes některé nedostatky je práce přínosem. Určitě si lze představit některé kapitoly pojaty jak více do hloubky, tak více do šířky. U analýzy nástrojů chybí více softwaru ke srovnání. U analýzy algoritmů taktéž nebyly jmenovány úplně všechny. Práce se prakticky omezila jen na jeden typ analýz (tj. ten, kde figurují kumulované náklady), z čehož vyplývá, že i zde je prostor k rozšíření. Ovšem nutno dodat, že všechny další analýzy z tohoto typu vycházejí. Autor se ovšem domnívá, že nic podstatného vzhledem k rozsahu práce v ní nechybí. Přínosem práce je určitě program, který je z hlediska možností jedinečný, stejně jako jeho výstupy. Taktéž popsání všech nástrojů ArcGIS, ILWIS a GRASS v dané oblasti autor považuje za přínos. Zajímavá je též zmínka o hybridním modelu dat (tj. rastr zkombinovaný s vektorem). Tento model by si určitě zasluhoval další pozornost.

Použité zdroje

BAYER, Tomáš. *Algoritmy v digitální kartografii*. 1. vyd. Praha: Karolinum, 2008. ISBN 978-80-246-1499-1

BAYER, Tomáš – SCHNEIDER, Michal. *Java pro geoinformatiky*. 2007. Materiál neprošel recenzním řízení

ČERNÝ, Jakub. *Základní grafové algoritmy* [online]. poslední úprava 2008 [cit. 2008-08-12]. <http://kam.mff.cuni.cz/~kuba/ka/min_cesta.pdf>

HALDEN, Derek - JONES, Peter – WIXEY, Sarah. *Accessibility Analysis Literature Review* [online]. Poslední úprava červen 2005 [cit. 2008-05-12]. <http://home.wmin.ac.uk/transport/download/SAMP_WP3_Accessibility_Modelling.pdf>

HUSDAL, Jan. *Network analysis - network versus vector - A comparison study* [online]. vznik 1999 [cit. 2008-08-14]. <http://husdal.typepad.com/blog/1999/10/network-analysi.html>

CHAN, Yupo. Measuring Spatial Separation: Distance, Time, Routing, and Accessibility. In *Location, Transport and Land-use* [online]. 1. vyd. Berlin: Springer Berlin Heidelberg, 2005. Kap. 3. <http://springerlink.metapress.com/content/j120jxqt0p407857/?p=880f6382033c408d859f1c2dfbc329ec&pi=4>. ISBN 978-3-540-26851-2. str.120 – 209

CHRISMAN, Nicholas. *Exploring Geographic Information Systems*. 1. vydání. 1997. ISBN 0-471-10842-1. str. 146-151

JIANG,B. – CLARAMUNT,C. - BATTY,M. *Geometric accessibility and geographic information: extending desktop GIS to space syntax* [online]. Computers, Environment and Urban Systems. 1999, Vol. 23.
<http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V9K-3WTNCRT-5-2P&_cdi=5901&_user=2788347&_orig=search&_coverDate=03%2F01%2F1999&_sk=999769997&view=c&wchp=dGLbVlz zSkWA&md5=e59ca4d7cb80642e321e0d233b0cb7f8&ie=/sdarticle.pdf>
ISSN 0198-9715. str. 127 – 146

LESTER, Patrick. *A* Pathfinding for Beginners* [online]. poslední úprava 14.7.2005 [cit. 2008-08-12].
<http://www.policyalmanac.org/games/aStarTutorial.htm>

LIU, Suxia - ZHU, Xuan. *An Integrated GIS Approach to Accessibility Analysis* [online]. Transactions in GIS. January 2004, Volume 8, Issue 1.
<http://www.blackwell-synergy.com/doi/pdf/10.1111/j.1467-9671.2004.00167.x?cookieSet=1> ISSN 1361-1682. str. 45 - 62

MILLER, Harvey J. *GIS Software for Measuring Space-Time Accessibility in Transportation Planning and Analysis* [online]. Geoinformatica. 2000
<http://springerlink.metapress.com/content/p4q12730314110u0/fulltext.pdf>
ISSN 1573-7624. str. 141 – 143

MITCHELL, Andy. *The ESRI® Guide to GIS Analysis: Volume 1: Geographic Patterns & Relationships*. 1. vydání. Redlands: Enviromental Systém Research Institute, Inc., 1999.
ISBN 1-879102-06-4. str. 118-147

SHEN, Qing. *Spatial Technologies, Accessibility, and The Social Construction of Urban Space* [online]. Computers, Environment and Urban Systems. 1998, Vol. 22, No. 5.
<http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V9K-3VGC4W5-N-1F&_cdi=5901&_user=2788347&_orig=search&_coverDate=09%2F01%2F1998&_sk=999779994&view=c&wchp=dGLzVlz-zSkWz&md5=b536435d8ac51b930874c0ff94dae6d6&ie=/sdarticle.pdf>
ISSN 0198-9715. str. 447 – 452

SMITH, Michael J. de – GOODCHILD, Michael F. – LONGLEY, Paul A.
Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools. [online].
2. vyd. 2007. <http://www.spatialanalysisonline.com/output/HandbookExtract.pdf>
ISBN 978-1906221-980

VAN ECK, Ritsema J.R. – DE JONG, T. *Accessibility analysis and spatial competition effects in the context of GIS-supported service location planning* [online]. Computers, Environment and Urban Systems. 1999, Vol.23.
<http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V9K-3WTNCRT-2-10&_cdi=5901&_user=2788347&_orig=search&_coverDate=03%2F01%2F1999&_sk=999769997&view=c&wchp=dGLbVlz-zSkzV&md5=0d782de801739639e83a06c46567d86d&ie=/sdarticle.pdf>

VÚGTK. Terminologický slovník VÚGTK [online]. 2005-2008 [cit. 2008-04-20].
<http://www.vugtk.cz/slovnik>

WEIBULL, Jörgen W. *An axiomatic approach to the measurement of accessibility* [online]. Regional Science and Urban Economics. December 1976, Vol. 6, Issue 4.
<http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V89-45PM74R-2&_user=2788347&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000053052&_version=1&_urlVersion=0&_userid=2788347&md5=a3a6a6a363012a525765da196b8e62dc> str. 357 - 358

WORBOYS, Michael F. *GIS: Computing Perspective*. 1. vydání, London: Taylor & Francis Ltd, 1995. ISBN 0-7484-0064-8. str. 230-238

Seznam příloh

Příloha 1 Výsledky testů časové náročnosti algoritmů

Tabulka časové náročnosti algoritmů nad vektorovými daty (čas v ms)

Pokus / analýza	50 U, 1 VB	50 U, 5 VB	500 U, 5 VB	500 U, 1 VB, B.	500 U, 1 VB, 20 hran	500 U, 50 VB	500 U, 50 VB, B.	5000 U, 1 VB, B.	50 U,1 VB, B.	5000 U, 1 VB
1	0,201	0,141	2,119	154,800	32,200	2,687	161,000	6168,000	1,515	20,795
2	0,196	0,144	1,155	163,100	25,364	2,632	167,600	6083,000	1,539	28,091
3	0,15	0,144	1,197	161,700	33,974	2,829	162,100	6310,000	1,590	22,713
4	0,142	0,144	1,217	167,140	32,006	2,375	163,500	6177,000	1,870	15,617
5	0,144	0,153	3,140	159,900	26,996	2,657	158,000	6438,000	1,553	21,730
6	0,138	0,146	1,167	166,100	22,213	2,831	163,900	6564,000	4,785	31,347
7	0,128	0,140	1,070	161,200	27,755	2,815	163,400	6232,000	1,697	20,601
8	0,134	0,137	1,131	161,800	27,798	3,143	160,700	6623,000	1,540	16,089
9	0,138	0,146	1,105	161,900	24,628	10,494	161,300	6224,000	1,497	18,311
10	0,138	0,159	1,155	160,500	23,738	2,673	177,000	7430,000	2,767	37,061
11	0,126	0,136	1,252	163,300	26,937	2,806	165,000	7880,000	1,609	19,693
12	0,131	0,887	1,500	173,400	32,962	3,256	164,300	7738,000	1,682	39,266
13	0,097	0,106	1,117	164,600	31,754	2,617	164,600	7562,000	1,564	26,285
14	0,106	0,123	1,420	163,000	27,062	2,577	159,000	6516,000	1,599	25,454
15	0,098	0,102	1,229	159,600	21,988	2,570	160,300	7109,000	1,588	16,411
Průměr	**0,138**	**0,187**	**1,398**	**162,803**	**27,825**	**3,264**	**163,447**	**6736,933**	**1,893**	**23,964**

Poznámka: U = uzlů; VB = výchozích bodů; B. = Bellman-Fordův algoritmus – jinak Dijkstrův algoritmus

Tabulka časové náročnosti algoritmů nad rastrovými daty (čas v ms)

Pokus / analýza	20 P, 4 S	20 P, 8 S	20 P, 16 S	200 P, 4 S	200 P, 8 S	200 P, 16 S	20 P, 8 S, 10 VB
1	6,561	8,811	4,715	397,700	889,600	2318,500	1,287
2	0,950	1,720	2,528	471,300	1165,700	3025,500	1,591
3	0,682	1,228	3,134	394,200	906,200	2380,400	1,505
4	0,771	1,067	2,386	434,304	1020,400	2642,500	1,889
5	0,530	1,180	2,750	336,765	788,800	2088,000	1,678
6	0,517	1,535	2,339	326,400	796,600	2016,800	1,984
7	0,462	0,950	2,272	434,700	1043,800	2664,000	1,730
8	0,449	0,978	2,642	321,700	748,700	1936,700	1,754
9	0,678	0,918	2,193	304,800	770,200	2003,700	1,982
10	7,141	1,276	3,052	616,800	1476,900	3769,500	1,693
11	0,420	0,840	1,985	383,500	897,400	2234,600	1,703
12	0,536	1,124	2,748	325,600	753,600	1971,000	1,858
13	0,502	0,838	1,989	342,700	777,500	2005,400	1,566
14	0,494	1,049	2,570	392,100	975,300	2469,900	1,628
15	0,612	0,898	2,137	333,520	794,600	2032,800	2,566
Průměr	**0,585**	**1,102**	**2,436**	**387,739**	**920,353**	**2370,620**	**1,761**

Poznámka: 20 P = 20×20 pixelů; 200 P = 200×200 pixelů; S = počet sousedících buněk; VB = výchozí body

Příloha 2

Kód programu „Analýzy prostorové dostupnosti“ v jazyce Java

```
package vektorova_sit;

import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SpinnerNumberModel;

public class Hlavni extends javax.swing.JFrame {

    private Sit sit;
    private Rastr rastr;
    private Uzel [] startovniUzly;
    private Pixel [] startovniPixely;
    private int pomer;
    private JFrame [] ramy=new JFrame[7];

    public Hlavni() {
        super("Analýzy prostorové dostupnosti");
        initComponents();
        setComponents();
    }

    public void setComponents() {
        jSpinner1.setModel(new SpinnerNumberModel(100,10,50000,1));
        jSpinner2.setModel(new SpinnerNumberModel(10,1,1000,1));
        jSpinner3.setModel(new SpinnerNumberModel(20,5,3000,1));
        jSpinner4.setModel(new SpinnerNumberModel(4,2,30,1));
        jSpinner5.setModel(new SpinnerNumberModel(1,1,100,1));
        jSpinner6.setModel(new SpinnerNumberModel(10,1,100,1));
    }

    private Pixel[] generujNahodneBody(Uzel[] startovniUzly) {
        Pixel [] vysledek=new Pixel[startovniUzly.length];
        for (int i=0;i<startovniUzly.length;i++) {
            vysledek[i]=new Pixel((int) (startovniUzly[i].getX())/pomer,
              (int) (startovniUzly[i].getY()/pomer),0);
        }
        return vysledek;
    }

    private void initComponents() {
        // Kód generovaný vývojovým prostředím není součástí přílohy
    }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
        if (ramy[0]!=null) {ramy[0].setVisible(false);}
        pomer=jSpinner6.getAccessibleContext().getAccessibleValue()
          .getCurrentAccessibleValue().intValue();
        ramy[0]=new JFrame("Zkonvertovaný rastr");
        JScrollPane panel=new JScrollPane();
        rastr=sit.konverze(pomer);
        panel.setViewportView(rastr.kresliBunky(false));
        ramy[0].add(panel);
```

```
        ramy[0].setMinimumSize(new Dimension(300,300));
        ramy[0].setVisible(true);
        jCheckBox1.setEnabled(true);
        jCheckBox2.setEnabled(true);
        jCheckBox3.setEnabled(true);
        jCheckBox4.setEnabled(true);
        jCheckBox5.setEnabled(true);
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        int pocetBodu=jSpinner5.getAccessibleContext().getAccessibleValue()
          .getCurrentAccessibleValue().intValue();
        if (sit!=null) {startovniUzly=sit.generujNahodneUzly(pocetBodu);}
        if ((rastr!=null)&&(sit!=null)) {startovniPixely
         =this.generujNahodneBody(startovniUzly);}
        if ((rastr!=null)&&(sit==null)) {startovniPixely
          =rastr.generujNahodneBody(pocetBodu);}
        for (int i=1;i<7;i++) {
            if (ramy[i]!=null) {
                ramy[i].setVisible(false);
            }
        }
        JScrollPane [] panely=new JScrollPane[7];
        ramy[6]=new JFrame("Časy analýz");
        JTextArea text=new JTextArea();
        ramy[6].add(text);
        ramy[6].setMinimumSize(new Dimension(300,300));

        if (jCheckBox1.isSelected()) {
            ramy[1]=new JFrame("4-směrný pohyb");
            panely[1]=new JScrollPane();
            Rastr vysledek=rastr.dijkstra(startovniPixely,0);
            panely[1].setViewportView(vysledek.kresliBunky(true));
            ramy[1].add(panely[1]);
            System.out.println(String.valueOf(vysledek.getCas()));
            text.append("4-smerny pohyb: "+String.valueOf
              (vysledek.getCas()/1000000)+" ms\n\n");
        }
        else ramy[1]=null;

        if (jCheckBox2.isSelected()) {
            ramy[2]=new JFrame("8-směrný pohyb");
            panely[2]=new JScrollPane();
            Rastr vysledek=rastr.dijkstra(startovniPixely,1);
            panely[2].setViewportView(vysledek.kresliBunky(true));
            ramy[2].add(panely[2]);
            System.out.println(String.valueOf(vysledek.getCas()));
            text.append("8-smerny pohyb: "+String.valueOf
              (vysledek.getCas()/1000000)+" ms\n\n");
        }
        else ramy[2]=null;

        if (jCheckBox3.isSelected()) {
            ramy[3]=new JFrame("16-směrný pohyb");
            panely[3]=new JScrollPane();
```

```
            Rastr vysledek=rastr.dijkstra(startovniPixely,2);
            panely[3].setViewportView(vysledek.kresliBunky(true));
            ramy[3].add(panely[3]);
            System.out.println(String.valueOf(vysledek.getCas()));
            text.append("16-smerny pohyb: "+String.valueOf
              (vysledek.getCas()/1000000)+" ms\n\n");
        }
        else ramy[3]=null;

        if (jCheckBox4.isSelected()) {
            ramy[4]=new JFrame("Dijkstrův algoritmus");
            panely[4]=new JScrollPane();
            VyslSit vysledek=sit.dijkstra(startovniUzly);
            panely[4].setViewportView(vysledek);
            ramy[4].add(panely[4]);
            System.out.println(String.valueOf(vysledek.getCas()));
            text.append("Dijkstruv algoritmus: "+String.valueOf
              (vysledek.getCas()/1000000)+" ms\n\n");
        }
        else ramy[4]=null;

        if (jCheckBox5.isSelected()) {
            ramy[5]=new JFrame("Bellman-Frodův algoritmus");
            panely[5]=new JScrollPane();
            VyslSit vysledek=sit.bellman(startovniUzly);
            panely[5].setViewportView(vysledek);
            ramy[5].add(panely[5]);
            System.out.println(String.valueOf(vysledek.getCas()));
            text.append("Bellman-Forduv algoritmus: "+String.valueOf
              (vysledek.getCas()/1000000)+" ms\n\n");
        }
        else ramy[5]=null;

        for (int i=1;i<6;i++) {
            if (ramy[i]!=null) {
                ramy[i].setMinimumSize(new Dimension(300,300));
                ramy[i].setVisible(true);
            }
        }
        ramy[6].setLocation(300,300);
        ramy[6].setVisible(true);
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        rastr=null;
        int velikost=jSpinner1.getAccessibleContext()
          .getAccessibleValue().getCurrentAccessibleValue().intValue();
        int hodnota=jSpinner2.getAccessibleContext()
          .getAccessibleValue().getCurrentAccessibleValue().intValue();
        int uzly=jSpinner3.getAccessibleContext()
          .getAccessibleValue().getCurrentAccessibleValue().intValue();
        int hran=jSpinner4.getAccessibleContext()
          .getAccessibleValue().getCurrentAccessibleValue().intValue();
        sit=new Sit(velikost,hran,uzly,hodnota);
        jScrollPane1.setViewportView(sit);
```

```
        jButton3.setEnabled(true);
        jButton4.setEnabled(true);
        jCheckBox1.setEnabled(false);
        jCheckBox1.setSelected(false);
        jCheckBox2.setEnabled(false);
        jCheckBox2.setSelected(false);
        jCheckBox3.setEnabled(false);
        jCheckBox3.setSelected(false);
        jCheckBox4.setEnabled(true);
        jCheckBox5.setEnabled(true);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        sit=null;
        int velikost=jSpinner1.getAccessibleContext()
          .getAccessibleValue().getCurrentAccessibleValue().intValue();
        int hodnota=jSpinner2.getAccessibleContext()
          .getAccessibleValue().getCurrentAccessibleValue().intValue();
        rastr=new Rastr(velikost,hodnota);
        rastr.generujRastr();
        jScrollPane1.setViewportView(rastr.kresliBunky(false));
        jButton3.setEnabled(true);
        jButton4.setEnabled(false);
        jCheckBox1.setEnabled(true);
        jCheckBox2.setEnabled(true);
        jCheckBox3.setEnabled(true);
        jCheckBox4.setEnabled(false);
        jCheckBox4.setSelected(false);
        jCheckBox5.setEnabled(false);
        jCheckBox5.setSelected(false);
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Hlavni().setVisible(true);
            }
        });
    }

    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JCheckBox jCheckBox1;
    private javax.swing.JCheckBox jCheckBox2;
    private javax.swing.JCheckBox jCheckBox3;
    private javax.swing.JCheckBox jCheckBox4;
    private javax.swing.JCheckBox jCheckBox5;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
```

```
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JSpinner jSpinner1;
    private javax.swing.JSpinner jSpinner2;
    private javax.swing.JSpinner jSpinner3;
    private javax.swing.JSpinner jSpinner4;
    private javax.swing.JSpinner jSpinner5;
    private javax.swing.JSpinner jSpinner6;
}
```

```
package vektorova_sit;

public class Hrana {

    private Uzel u1;
    private Uzel u2;
    private int atribut;
    private int indexHrany;
    private static int pocet=0;
    private static Hrana[] hrany;

    public Hrana(Uzel u1,Uzel u2,int max) {
        this.u1=u1;
        this.u2=u2;
        indexHrany=pocet;
        hrany[pocet]=this;
        pocet++;
        atribut=(int) (Math.random()*max)+1;
        u2.nastavHranu(this);
    }

    public int delkaHrany() {
        double a=Math.pow(u1.getX()-u2.getX(),2)
          +Math.pow(u1.getY()-u2.getY(),2);
        int x=(int) (Math.sqrt(a));
        return x;
    }

    public Uzel getUzel1() {
        return this.u1;
    }

    public Uzel getUzel2() {
        return this.u2;
    }

    public static Hrana getHrana(int index_hr) {
        return hrany[index_hr];
    }

    public static int getPocetHran() {
        return pocet;
    }

    public int getAtribut() {
        return atribut;
    }

    public static void nulujIndex() {
        pocet=0;
    }

    public static void initHrany() {
        hrany=new Hrana[(Uzel.getPocetUzlu()*Uzel.getPocetUzlu()-1)/2];
    }
}
```

```java
package vektorova_sit;

public class Uzel {

    private int x;
    private int y;
    private int index;
    private int atribut;
    private Hrana [] hrany;
    private int pocetHran;
    private int maxPocetHran;
    private boolean pridano;
    private static int pocetUzlu;

    public Uzel(int x,int y,int index,int max_hran,int atribut) {
        pocetHran=0;
        this.x=x;
        this.y=y;
        this.index=index;
        this.maxPocetHran=max_hran;
        this.atribut=atribut;
        hrany=new Hrana[max_hran];
        pridano=false;
    }

    public static void setPocetUzlu(int pocet) {
        pocetUzlu=pocet;
    }

    public boolean isConnected(Uzel u2) {
        boolean connected=false;
        for (int i=0;i<pocetHran;i++) {
            if (this.hrany[i].getUzel1().getIndex()==u2.getIndex()) {
                connected=true;
            }
            if (this.hrany[i].getUzel2().getIndex()==u2.getIndex()) {
                connected=true;
            }
        }
        if (this.getIndex()==u2.getIndex()) {
            connected=true;
        }
        return connected;
    }

    public void pridejHranu(Uzel u2,int max) {
        hrany[pocetHran]=new Hrana(this,u2,max);
        pocetHran++;
    }

    public void nastavHranu(Hrana h) {
        hrany[pocetHran]=h;
        pocetHran++;
```

```
}

public void orezHrany() {
    Hrana [] nove=new Hrana [pocetHran];
    for (int i=0;i<nove.length;i++) {
        nove[i]=hrany[i];
    }
    hrany=nove;
}

public Uzel [] najdiSousedy() {
    Uzel u1,u2;
    int j=0;
    Uzel [] sousedi=new Uzel[hrany.length];
    Uzel [] sousedi2;
    for (int i=0;i<hrany.length;i++) {
        u1=hrany[i].getUzel1();
        u2=hrany[i].getUzel2();
        if ((u1==this)&&(!u2.getPridano())) {
            sousedi[j]=u2;
            j++;
        }
        if ((u2==this)&&(!u1.getPridano())) {
            sousedi[j]=u1;
            j++;
        }
    }
    if (j!=0) {
        sousedi2=new Uzel[j];

        for (int i=0;i<j;i++) {
            sousedi2[i]=sousedi[i];
        }
    } else {
        sousedi2=new Uzel[1];
    }

    return sousedi2;
}

public int getIndex() {
    return this.index;
}

public int getMaxHran() {
    return this.maxPocetHran;
}

public int getPocetHran() {
    return this.pocetHran;
}

public int getX() {
    return this.x;
}
```

```
    public int getY() {
        return this.y;
    }

    public Hrana [] getHrany() {
        return hrany;
    }

    public int getAtribut() {
        return atribut;
    }

    public void setAtribut(int atr) {
        this.atribut=atr;
    }

    public static int getPocetUzlu() {
        return pocetUzlu;
    }

    public boolean getPridano() {
        return pridano;
    }

    public void setPridano(boolean prid) {
        this.pridano=prid;
    }

    public static void nulujPocetUzlu() {
        pocetUzlu=0;
    }
}
```

```java
package vektorova_sit;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.TreeMap;
import java.util.TreeSet;
import javax.swing.JComponent;

public class Sit extends JComponent {

    private Uzel [] uzly;
    private int souradMax;
    private int hranyStred;
    private int pocetUzlu;
    private int max;
    private ArrayList<Uzel> fronta;

    public Sit(int sourad_max,int hrany_stred,int pocet_uzlu,int max) {
        this.souradMax=sourad_max;
        this.hranyStred=hrany_stred;
        this.pocetUzlu=pocet_uzlu;
        this.max=max;

        Uzel.setPocetUzlu(pocet_uzlu); //vygenerování uzlů
        this.generujUzly();

        Hrana.initHrany(); // vygenerování hran
        Hrana.nulujIndex();
        this.generujHrany();

        this.setPreferredSize(new Dimension(sourad_max,sourad_max));
    }

    public void paintComponent(Graphics g) {
        int x,y,x1,x2,y1,y2;
        double atribut;

        g.setColor(Color.gray);
        for  (int i=0;i<Hrana.getPocetHran();i++) {
            x1=Hrana.getHrana(i).getUzel1().getX();
            x2=Hrana.getHrana(i).getUzel2().getX();
            y1=Hrana.getHrana(i).getUzel1().getY();
            y2=Hrana.getHrana(i).getUzel2().getY();
            g.drawLine(x1,y1,x2,y2);
        }

        g.setColor(Color.BLACK);
        for (int i=0;i<uzly.length;i++) {
            x=uzly[i].getX();
            y=uzly[i].getY();
            g.fillOval(x-3,y-3,6,6);
```

```
        }

        g.setColor(Color.RED);
        for  (int i=0;i<Hrana.getPocetHran();i++) {
            x1=Hrana.getHrana(i).getUzel1().getX();
            x2=Hrana.getHrana(i).getUzel2().getX();
            y1=Hrana.getHrana(i).getUzel1().getY();
            y2=Hrana.getHrana(i).getUzel2().getY();
            atribut=Hrana.getHrana(i).getAtribut();
            g.setColor(Color.RED);
            g.drawString(String.valueOf(atribut),(x1+x2)/2-6,(y1+y2)/2);
            g.setColor(Color.BLUE);
            g.drawString(String.valueOf(atribut*Hrana.getHrana(i)
              .delkaHrany()),(x1+x2)/2-6,(y1+y2)/2+12);
        }
    }

    public void generujUzly() {
        uzly=new Uzel[pocetUzlu];
        int x,y,max_hran;
        for (int i=0;i<pocetUzlu;i++) {
            x=(int) (Math.random()*souradMax);
            y=(int) (Math.random()*souradMax);
            max_hran=this.generujPocetHran();
            uzly[i]=new Uzel(x,y,i,max_hran,0);
        }
    }

    public void generujHrany() {
        Uzel [] nejblizsi;
        for (int i=0;i<pocetUzlu-1;i++) {
          nejblizsi=this.najdiNejblizsi(uzly[i].getMaxHran()
                  -uzly[i].getPocetHran(),uzly[i]);
          for (int j=0;j<nejblizsi.length;j++) {
              uzly[i].pridejHranu(nejblizsi[j],max);
          }
          if (uzly[i].getMaxHran()!=uzly[i].getPocetHran()) {
              uzly[i].orezHrany();
          }
        }
    }

    public Uzel [] najdiNejblizsi(int pocet,Uzel u) {
        double [][] vzdalenosti;
        vzdalenosti=this.getVzdalenosti(u);
        Uzel [] vrac_uzly=new Uzel[Math.min(vzdalenosti.length,pocet)];
        for (int i=0;i<Math.min(vzdalenosti.length,pocet);i++) {
            vrac_uzly[i]=uzly[(int)vzdalenosti[i][1]];
        }
        return vrac_uzly;
    }

    public double [][] getVzdalenosti(Uzel u) {
        int rozdil_x,rozdil_y;
        int k=0;
```

```
        double [][] vzdalenost=new double[pocetUzlu][2];
        double [][] serazena_vzd;
        for (int i=0;i<pocetUzlu;i++) {
            if ((!u.isConnected(uzly[i]))&&(uzly[i].getMaxHran()
              !=uzly[i].getPocetHran()))
            {
                rozdil_x=uzly[i].getX()-u.getX();
                rozdil_y=uzly[i].getY()-u.getY();
                vzdalenost[k][0]=
                  Math.sqrt(rozdil_x*rozdil_x+rozdil_y*rozdil_y);
                vzdalenost[k][1]=i;
                k++;
            }
        }
        serazena_vzd=new double[k][2];
        double min;
        int index,j,index2;
        for (int i=0;i<k;i++) {
            min=vzdalenost[i][0];
            index=(int)vzdalenost[i][1];
            index2=i;
            for (j=i+1;j<k;j++) {
                if (vzdalenost[j][0]<min) {
                    min=vzdalenost[j][0];
                    index=(int)vzdalenost[j][1];
                    index2=j;
                }
            }
            serazena_vzd[i][0]=min;
            serazena_vzd[i][1]=index;
            vzdalenost[index2][0]=vzdalenost[i][0];
            vzdalenost[index2][1]=vzdalenost[i][1];
        }
        return serazena_vzd;
    }

    public int generujPocetHran() {
        int pocet=0;
        if (hranyStred%2!=0) {
            pocet=(int) (Math.random()*hranyStred)+1+(int) hranyStred/2;
        }
        else {
            pocet=(int) (Math.random()*(hranyStred-1))+1
              +(int) hranyStred/2;
        }
        return pocet;
    }

    public VyslSit dijkstra(Uzel[] startovni) {
        this.nulujUzly();
        double cas1=System.nanoTime();
        VyslSit vysledek=new VyslSit(souradMax);
        fronta=new ArrayList<Uzel> ();

        inicializaceVychozich(startovni,vysledek);
```

```
        while (!fronta.isEmpty()) {
            this.pridejDoVysledne(fronta.get(0),vysledek);
        }
        double cas2=System.nanoTime();
        vysledek.setCas(cas2-cas1);

        return vysledek;
    }

    public void inicializaceVychozich(Uzel[]startovni,VyslSit vysledek) {
        for (int i=0;i<startovni.length;i++) {
            startovni[i].setAtribut(0);
            startovni[i].setPridano(true);
            vysledek.pridej(new Uzel(startovni[i].getX(),
              startovni[i].getY(),startovni[i].getIndex(),0,0));
            Uzel [] sousedi=startovni[i].najdiSousedy();
            if (sousedi[0]!=null) {
                for (int j=0;j<sousedi.length;j++) {
                    this.pridejDoFronty(startovni[i],sousedi[j]);
                }
            }
        }
    }

    public void pridejDoVysledne(Uzel u,VyslSit vysledek) {
        vysledek.pridej(new Uzel(u.getX(),u.getY(),u.getIndex(),
          0,u.getAtribut()));
        u.setPridano(true);
        fronta.remove(0);
        Uzel [] sousedi=u.najdiSousedy();
        if (sousedi[0]!=null) {
            for (int i=0;i<sousedi.length;i++) {
                this.pridejDoFronty(u,sousedi[i]);
            }
        }
    }

    public void pridejDoFronty(Uzel vychozi,Uzel pridavany) {
        Hrana hrana=vychozi.getHrany()[0];
        Uzel uz1,uz2;
        int j=0;
        for (int i=0;i<vychozi.getHrany().length;i++) {//nalezení hrany
            uz1=vychozi.getHrany()[i].getUzel1();
            uz2=vychozi.getHrany()[i].getUzel2();
            if (((uz1==vychozi)&&(uz2==pridavany))
              ||((uz1==pridavany)&&(uz2==vychozi)))
            {
                hrana=vychozi.getHrany()[i];
            }
        }
        if (!fronta.contains(pridavany)) { //první navštívení uzlu
            pridavany.setAtribut(vychozi.getAtribut()
              +hrana.getAtribut()*hrana.delkaHrany());
            j=0;
            if (!fronta.isEmpty()) {
```

```
                while ((j<fronta.size())&&(fronta.get(j).getAtribut()
                  <pridavany.getAtribut()))
                {
                    j++;
                }
            }
            fronta.add(j,pridavany);
        }
        else { //opakované navštívení uzlu
            if (pridavany.getAtribut()>(vychozi.getAtribut()
              +hrana.getAtribut()*hrana.delkaHrany()))
            {
                pridavany.setAtribut((vychozi.getAtribut()
                  +hrana.getAtribut()*hrana.delkaHrany()));
                j=0;
                if (!fronta.isEmpty()) {
                    while ((j<fronta.size())&&(fronta.get(j).getAtribut()
                      <pridavany.getAtribut()))
                    {
                        j++;
                    }
                }
                fronta.add(j,pridavany);
            }
        }
    }

    public Rastr konverze(int pomer) {
        Uzel u1,u2;
        float smernice;
        float rozdilX,rozdilY;
        double y=0;
        Rastr vysledek=new Rastr(souradMax/pomer,max);
        for (int i=0;i<Hrana.getPocetHran();i++) {
            // převedení postupně po hranách
            if (Hrana.getHrana(i).getUzel1().getX()>Hrana.getHrana(i)
              .getUzel2().getX())
            {
                u2=Hrana.getHrana(i).getUzel1();
                u1=Hrana.getHrana(i).getUzel2();
            }
            else {
                u1=Hrana.getHrana(i).getUzel1();
                u2=Hrana.getHrana(i).getUzel2();
            }
            if ((int) (u1.getX()/pomer)==(int) (u2.getX()/pomer)) {
                //svislý vektor
                if (u1.getY()>u2.getY()) {
                    for (int j=(int) (u2.getY()/pomer);j<
                      (int) (u1.getY()/pomer)+1;j++)
                    {
                        vysledek.pridej(new Pixel((int) (u1.getX()/pomer),
                          j,(int)(Hrana.getHrana(i).getAtribut()*pomer)));
                    }
                }
```

```
        else {
            for (int j=(int) (u1.getY()/pomer);j<
              (int) (u2.getY()/pomer)+1;j++)
            {
                vysledek.pridej(new Pixel((int) (u1.getX()/pomer),
                  j,(int)(Hrana.getHrana(i).getAtribut()*pomer)));
            }
        }

    }
    else { //jakýkoliv jiný vektor
       rozdilY=u2.getY()-u1.getY();
       rozdilX=u2.getX()-u1.getX();
       smernice=rozdilY/rozdilX;
       y=(int) (u1.getY()/pomer);
       if (smernice>=0) {
           for (int j=(int) (u1.getX()/pomer);j<(int)
             (u2.getX()/pomer)+1;j++)
           {
               if ((j==(int) (u1.getX()/pomer))
                 ||(j==(int) (u2.getX()/pomer)))
               {
                   for (int k=(int) Math.round(y);
                     k<=Math.round(y+smernice/2);k++)
                   {
                       if ((k>=0)&&(k<(int) (souradMax/pomer))){
                           vysledek.pridej(new Pixel(j,k,(int)
                             (Hrana.getHrana(i).getAtribut()
                             *pomer)));
                       }
                   }
                   y=y+smernice/2;
               }
               else {
                   for (int k=(int) Math.round(y);
                     k<=Math.round(y+smernice);k++)
                   {
                       if ((k>=0)&&(k<(int) (souradMax/pomer))){
                           vysledek.pridej(new Pixel(j,k,(int)
                             (Hrana.getHrana(i).getAtribut()
                             *pomer)));
                       }
                   }
                   y=y+smernice;
               }
           }
       }
       else {
           for (int j=(int) (u1.getX()/pomer);j<
             (int) (u2.getX()/pomer)+1;j++)
           {
               if ((j==(int) (u1.getX()/pomer))
                 ||(j==(int) (u2.getX()/pomer)))
               {
                   for (int k=(int) Math.round(y);
```

```
                        k>=Math.round(y+smernice/2);k--)
                    {
                        if ((k>=0)&&(k<(int) (souradMax/pomer))){
                            vysledek.pridej(new Pixel(j,k,(int)
                              (Hrana.getHrana(i).getAtribut()
                              *pomer)));
                        }
                    }
                    y=y+smernice/2;
                }
                else {
                    for (int k=(int) Math.round(y);
                      k>=Math.round(y+smernice);k--)
                    {
                        if ((k>=0)&&(k<(int) (souradMax/pomer))){
                            vysledek.pridej(new Pixel(j,k,(int)
                              (Hrana.getHrana(i).getAtribut()
                              *pomer)));
                        }
                    }
                    y=y+smernice;
                }
            }
        }
    }
    for (int i=0;i<souradMax/pomer;i++) {
        for (int j=0;j<souradMax/pomer;j++) {
            if (vysledek.getPixelAt(i,j)==null) {
                vysledek.pridej(new Pixel(i,j,-1));
            }
        }
    }
    return vysledek;
}

public VyslSit bellman(Uzel[]startovni) {
    this.nulujUzly();
    double cas1=System.nanoTime();
    VyslSit vysledek=new VyslSit(souradMax);
    for (int i=0;i<startovni.length;i++) {
        startovni[i].setAtribut(1);
    }
    for (int i=0;i<pocetUzlu;i++) {
        if (uzly[i].getAtribut()==0) {
            uzly[i].setAtribut(10000000);
        }
        if (uzly[i].getAtribut()==1) {
            uzly[i].setAtribut(0);
        }
    }

    for (int i=0;i<pocetUzlu-1;i++) {
        for (int j=0;j<Hrana.getPocetHran();j++) {
```

```
                if (Hrana.getHrana(j).getUzel1().getAtribut()>
                  (Hrana.getHrana(j).getUzel2().getAtribut()
                  +Hrana.getHrana(j).getAtribut()
                  *Hrana.getHrana(j).delkaHrany()))
                {
                    Hrana.getHrana(j).getUzel1().setAtribut
                      ((Hrana.getHrana(j).getUzel2().getAtribut()
                      +Hrana.getHrana(j).getAtribut()
                      *Hrana.getHrana(j).delkaHrany()));
                }
                if (Hrana.getHrana(j).getUzel2().getAtribut()>
                  (Hrana.getHrana(j).getUzel1().getAtribut()
                  +Hrana.getHrana(j).getAtribut()*
                  Hrana.getHrana(j).delkaHrany()))
                {
                    Hrana.getHrana(j).getUzel2().setAtribut
                      ((Hrana.getHrana(j).getUzel1().getAtribut()+
                      Hrana.getHrana(j).getAtribut()*
                      Hrana.getHrana(j).delkaHrany()));
                }
            }
        }

        for (int i=0;i<pocetUzlu;i++) {
            vysledek.pridej(new Uzel(uzly[i].getX(),uzly[i].getY(),
             i,0,uzly[i].getAtribut()));
        }
        double cas2=System.nanoTime();
        vysledek.setCas(cas2-cas1);
        return vysledek;
    }

    public Uzel [] generujNahodneUzly(int pocet) {
        int a;
        Uzel [] nahodne=new Uzel[pocet];
        for (int i=0;i<pocet;i++) {
            a=(int) (Math.random()*Uzel.getPocetUzlu());
            nahodne[i]=uzly[a];
        }
        return nahodne;
    }

    public void nulujUzly() {
        for (int i=0;i<Uzel.getPocetUzlu();i++) {
            uzly[i].setPridano(false);
            uzly[i].setAtribut(0);
        }
    }
}
```

```
package vektorova_sit;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JComponent;

public class VyslSit extends JComponent{

    private Uzel[] uzly=new Uzel[Uzel.getPocetUzlu()];
    private double cas;

    public VyslSit(int sourad_max) {
        this.setPreferredSize(new Dimension(sourad_max,sourad_max));
    }

    public void paintComponent(Graphics g) {
       int x,y,x1,x2,y1,y2;
       int atribut;

       g.setColor(Color.gray);
       for  (int i=0;i<Hrana.getPocetHran();i++) {
            x1=Hrana.getHrana(i).getUzel1().getX();
            x2=Hrana.getHrana(i).getUzel2().getX();
            y1=Hrana.getHrana(i).getUzel1().getY();
            y2=Hrana.getHrana(i).getUzel2().getY();
            g.drawLine(x1,y1,x2,y2);
        }

       g.setColor(Color.BLACK);
        for (int i=0;i<uzly.length;i++) {
           x=uzly[i].getX();
           y=uzly[i].getY();
           g.fillOval(x-3,y-3,6,6);
        }

       g.setColor(Color.RED);
       for (int i=0;i<uzly.length;i++) {
           x=uzly[i].getX();
           y=uzly[i].getY();
           g.drawString(String.valueOf(uzly[i].getAtribut()),x,y);
       }
    }
    public void pridej(Uzel u) {
        uzly[u.getIndex()]=u;
    }

    public void setCas(double cas) {
            this.cas=cas;
    }

    public double getCas() {
        return this.cas;
    }
}
```

```java
package vektorova_sit;

public class Pixel {
    private int x;
    private int y;
    private int hodnota;
    private int kumul;
    private boolean pridano;

    public Pixel(int x,int y,int hodnota) {
        this.x=x;
        this.y=y;
        this.hodnota=hodnota;
        this.pridano=false;
    }

    public void setPridano(boolean prid) {
        this.pridano=prid;
    }

    public void setHodnota(int hodnota) {
        this.hodnota=hodnota;
    }

    public void setKumul(int hodnota) {
        this.kumul=hodnota;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getHodnota() {
        return hodnota;
    }

    public int getKumul() {
        return kumul;
    }

    public boolean getPridano() {
        return pridano;
    }
}
```

```
package vektorova_sit;

import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.imageio.ImageIO;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class Rastr extends JComponent{

    private Pixel[][] bunky;
    private int velikost;
    private int max;
    private ArrayList<Pixel> fronta;
    private double cas;

    public Rastr(int velikost,int max) {
        this.velikost=velikost;
        this.bunky=new Pixel[velikost][velikost];
        this.max=max;
    }
    public void generujRastr() {
        for (int i=0;i<velikost;i++) {
            for (int j=0;j<velikost;j++) {
                bunky[i][j]=new Pixel(i,j,(int) (Math.random()*max)+1);
            }
        }
    }

    public JPanel kresliBunky(boolean kumul) {
        JPanel jPanel=new JPanel();
        GridBagLayout l=new GridBagLayout();
        GridBagConstraints c=new GridBagConstraints();
        jPanel.setLayout(l);
        if (velikost<130) {
            for (int i=0;i<velikost;i++) {
                for (int j=0;j<velikost;j++) {
                    c.gridx=i;
                    c.gridy=j;
                    if (kumul) {
                        jPanel.add(new JLabel(" "+String.valueOf(
                          bunky[i][j].getKumul())+" "),c);
                    } else {
                        jPanel.add(new JLabel(" "+String.valueOf(
                          bunky[i][j].getHodnota())+" "),c);
                    }
```

```
            }
        }
    }
    else {
        jPanel.add(new JLabel("Analyza probehla," +
          " z pametovych duvodu nedochazi k zobrazeni"));
    }
    return jPanel;
}


public Pixel[] generujNahodneBody(int pocet) {
    int x,y;
    Pixel[] nahodne=new Pixel[pocet];
    for (int i=0;i<pocet;i++) {
        x=(int) (Math.random()*velikost);
        y=(int) (Math.random()*velikost);
        nahodne[i]=bunky[x][y];
    }
    return nahodne;
}

public Rastr dijkstra(Pixel[]startovni,int pohyb)
{
    double cas1=System.nanoTime();
    int pridano=startovni.length; //počet již uzavřených buněk
    Rastr vysledek=new Rastr(velikost,max);
    fronta=new ArrayList<Pixel> ();

    inicializaceVychozich(startovni,vysledek,pohyb);
    while (!fronta.isEmpty()) { // postupný průběh algoritmu
        this.pridejDoVysledne(fronta.get(0),vysledek,pohyb);
        pridano++;
    }

    if (pridano!=velikost*velikost) {
        //pro analýzy nad konvertovanými rastry
        konvertovaneRastry(vysledek,velikost);
    }

    this.nulujPridano(); // příprava pro další analýzy
    double cas2=System.nanoTime();
    vysledek.cas=cas2-cas1;

    return vysledek;
}

public void inicializaceVychozich(Pixel[]startovni,
  Rastr vysledek, int pohyb)
{
    for (int i=0;i<startovni.length;i++) {
        // inicializace startovních bodů
        startovni[i].setPridano(true);
        startovni[i].setKumul(0);
        vysledek.pridej(startovni[i]);
```

```
            Pixel[] sousedi=this.najdiSousedy(startovni[i],pohyb);
            for (int j=0;j<sousedi.length;j++) {
                this.pridejDoFronty(startovni[i],sousedi[j]);

            }
        }
    }

    public void pridejDoVysledne(Pixel p,Rastr vysledek,int pohyb) {
        vysledek.pridej(p);
        p.setPridano(true);
        fronta.remove(0);
        Pixel [] sousedi=this.najdiSousedy(p,pohyb);
        for (int i=0;i<sousedi.length;i++) {
            this.pridejDoFronty(p,sousedi[i]);
        }
    }

    public Pixel[] najdiSousedy(Pixel p,int pohyb) {
        Pixel[]sousedi=new Pixel[16];
        int x,y;
        int i=0;
        boolean xmax=false;boolean xmin=false;
        boolean ymax=false;boolean ymin=false;
        boolean xmax2=false;boolean xmin2=false;
        boolean ymax2=false;boolean ymin2=false;
        x=p.getX();
        y=p.getY();
        if (x==(velikost-1)) {xmax=true;xmax2=true;}
        if (x==0) {xmin=true;xmin2=true;}
        if (y==(velikost-1)) {ymax=true;ymax2=true;}
        if (y==0) {ymin=true;ymin2=true;}
        if (x==(velikost-2)) {xmax2=true;}
        if (x==1) {xmin2=true;}
        if (y==(velikost-2)) {ymax2=true;}
        if (y==1) {ymin2=true;}

        if ((!xmax)&&(!bunky[x+1][y].getPridano())&&(bunky[x+1][y]
          .getHodnota()!=-1)) {sousedi[i]=bunky[x+1][y];i++;}
        if ((!ymin)&&(!bunky[x][y-1].getPridano())&&(bunky[x][y-1]
          .getHodnota()!=-1)) {sousedi[i]=bunky[x][y-1];i++;}
        if ((!xmin)&&(!bunky[x-1][y].getPridano())&&(bunky[x-1][y]
          .getHodnota()!=-1)) {sousedi[i]=bunky[x-1][y];i++;}
        if ((!ymax)&&(!bunky[x][y+1].getPridano())&&(bunky[x][y+1]
          .getHodnota()!=-1)) {sousedi[i]=bunky[x][y+1];i++;}

        if (pohyb>0) {
            if ((!xmax)&&(!ymax)&&(!bunky[x+1][y+1].getPridano())&&
              (bunky[x+1][y+1].getHodnota()!=-1))
              {sousedi[i]=bunky[x+1][y+1];i++;}
            if ((!xmax)&&(!ymin)&&(!bunky[x+1][y-1].getPridano())&&
              (bunky[x+1][y-1].getHodnota()!=-1))
              {sousedi[i]=bunky[x+1][y-1];i++;}
            if ((!xmin)&&(!ymin)&&(!bunky[x-1][y-1].getPridano())&&
              (bunky[x-1][y-1].getHodnota()!=-1))
```

```
            {sousedi[i]=bunky[x-1][y-1];i++;}
        if ((!xmin)&&(!ymax)&&(!bunky[x-1][y+1].getPridano())&&
          (bunky[x-1][y+1].getHodnota()!=-1))
          {sousedi[i]=bunky[x-1][y+1];i++;}

        if (pohyb>1) {
            if ((!xmax2)&&(!ymax)&&(!bunky[x+2][y+1].getPridano())&&
              (bunky[x+2][y+1].getHodnota()!=-1)&&
              (bunky[x+1][y+1].getHodnota()!=-1)&&
              (bunky[x+1][y].getHodnota()!=-1))
              {sousedi[i]=bunky[x+2][y+1];i++;}
            if ((!xmax2)&&(!ymin)&&(!bunky[x+2][y-1].getPridano())&&
              (bunky[x+2][y-1].getHodnota()!=-1)&&
              (bunky[x+1][y-1].getHodnota()!=-1)&&
              (bunky[x+1][y].getHodnota()!=-1))
              {sousedi[i]=bunky[x+2][y-1];i++;}
            if ((!xmax)&&(!ymax2)&&(!bunky[x+1][y+2].getPridano())&&
              (bunky[x+1][y+2].getHodnota()!=-1)&&
              (bunky[x+1][y+1].getHodnota()!=-1)&&
              (bunky[x][y+1].getHodnota()!=-1))
              {sousedi[i]=bunky[x+1][y+2];i++;}
            if ((!xmax)&&(!ymin2)&&(!bunky[x+1][y-2].getPridano())&&
              (bunky[x+1][y-2].getHodnota()!=-1)&&
              (bunky[x+1][y-1].getHodnota()!=-1)&&
              (bunky[x][y-1].getHodnota()!=-1))
              {sousedi[i]=bunky[x+1][y-2];i++;}
            if ((!xmin)&&(!ymax2)&&(!bunky[x-1][y+2].getPridano())&&
              (bunky[x-1][y+2].getHodnota()!=-1)&&
              (bunky[x-1][y+1].getHodnota()!=-1)&&
              (bunky[x][y+1].getHodnota()!=-1))
              {sousedi[i]=bunky[x-1][y+2];i++;}
            if ((!xmin)&&(!ymin2)&&(!bunky[x-1][y-2].getPridano())&&
              (bunky[x-1][y-2].getHodnota()!=-1)&&
              (bunky[x-1][y-1].getHodnota()!=-1)&&
              (bunky[x][y-1].getHodnota()!=-1))
              {sousedi[i]=bunky[x-1][y-2];i++;}
            if ((!xmin2)&&(!ymax)&&(!bunky[x-2][y+1].getPridano())&&
              (bunky[x-2][y+1].getHodnota()!=-1)&&
              (bunky[x-1][y+1].getHodnota()!=-1)&&
              (bunky[x-1][y].getHodnota()!=-1))
               {sousedi[i]=bunky[x-2][y+1];i++;}
            if ((!xmin2)&&(!ymin)&&(!bunky[x-2][y-1].getPridano())&&
              (bunky[x-2][y-1].getHodnota()!=-1)&&
              (bunky[x-1][y-1].getHodnota()!=-1)&&
              (bunky[x-1][y].getHodnota()!=-1))
              {sousedi[i]=bunky[x-2][y-1];i++;}
        }
    }
    Pixel[]sousedi2=new Pixel[i];
    for (int j=0;j<sousedi2.length;j++) {
        sousedi2[j]=sousedi[j];
    }
    return sousedi2;
}
```

```
public void pridejDoFronty(Pixel vychozi,Pixel pridavany) {
    int rozdilX,rozdilY,j;
    double delka,pruchod=0;
    rozdilX=pridavany.getX()-vychozi.getX();
    rozdilY=pridavany.getY()-vychozi.getY();
    delka=Math.sqrt((rozdilX*rozdilX)+(rozdilY*rozdilY));
    if (delka<1.05) { //přímý směr
        pruchod=(vychozi.getHodnota()+pridavany.getHodnota())/2;
    }
    if ((delka>1.05)&&(delka<1.5)) { //diagonála
        pruchod=Math.sqrt(2)*(vychozi.getHodnota()
          +pridavany.getHodnota())/2;
    }
    if (delka>1.5) {
        if ((rozdilX==2)&&(rozdilY==1)) { //pohyb jezdce
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()+1]
              [vychozi.getY()+1].getHodnota()+bunky[vychozi
              .getX()+1][vychozi.getY()].getHodnota())/4;
        }
        if ((rozdilX==2)&&(rozdilY==-1)) {
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()+1]
              [vychozi.getY()-1].getHodnota()+bunky[vychozi
              .getX()+1][vychozi.getY()].getHodnota())/4;
        }
        if ((rozdilX==1)&&(rozdilY==2)) {
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()+1]
              [vychozi.getY()+1].getHodnota()+bunky[vychozi
              .getX()][vychozi.getY()+1].getHodnota())/4;
        }
        if ((rozdilX==1)&&(rozdilY==-2)) {
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()+1]
              [vychozi.getY()-1].getHodnota()+bunky[vychozi
              .getX()][vychozi.getY()-1].getHodnota())/4;
        }
        if ((rozdilX==-1)&&(rozdilY==2)) {
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()]
              [vychozi.getY()+1].getHodnota()+bunky[vychozi
              .getX()-1][vychozi.getY()+1].getHodnota())/4;
        }
        if ((rozdilX==-1)&&(rozdilY==-2)) {
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()-1]
              [vychozi.getY()-1].getHodnota()+bunky[vychozi
              .getX()][vychozi.getY()-1].getHodnota())/4;
        }
        if ((rozdilX==-2)&&(rozdilY==1)) {
            pruchod=Math.sqrt(5)*(vychozi.getHodnota()
              +pridavany.getHodnota()+bunky[vychozi.getX()-1]
              [vychozi.getY()+1].getHodnota()+bunky[vychozi
              .getX()-1][vychozi.getY()].getHodnota())/4;
```

```
            }
            if ((rozdilX==-2)&&(rozdilY==-1)) {
                pruchod=Math.sqrt(5)*(vychozi.getHodnota()
                  +pridavany.getHodnota()+bunky[vychozi.getX()-1]
                  [vychozi.getY()-1].getHodnota()+bunky[vychozi
                  .getX()-1][vychozi.getY()].getHodnota())/4;
            }
        }
    if (!fronta.contains(pridavany)) { //první přidání do fronty
            pridavany.setKumul(vychozi.getKumul()+(int) pruchod);
            j=0;
            if (!fronta.isEmpty()) {
                while ((j<fronta.size())&&(fronta.get(j).getKumul()
                  <pridavany.getKumul()))
                {
                    j++;
                }
            }
            fronta.add(j,pridavany);
        }
        else { //buňka už ve frontě je
            if (pridavany.getKumul()>(vychozi.getKumul()+(int) pruchod))
            { //úprava hodnot
                pridavany.setKumul(vychozi.getKumul()+(int) pruchod);
                j=0;
                fronta.remove(pridavany);
                if (!fronta.isEmpty()) {
                    while ((j<fronta.size())&&(fronta.get(j).getKumul()
                      <pridavany.getKumul()))
                    {
                        j++;
                    }}
                fronta.add(j,pridavany);
            }
        }
    }

    public void konvertovaneRastry(Rastr vysledek,int velikost) {
        for (int i=0;i<velikost;i++) {
            for (int j=0;j<velikost;j++) {
                if (vysledek.bunky[i][j]==null) {
                    vysledek.bunky[i][j]=new Pixel(i,j,0);
                    vysledek.bunky[i][j].setKumul(-1);
                }
            }
        }
    }

    public void pridej(Pixel p) {
        this.bunky[p.getX()][p.getY()]=p;
    }

    public Pixel getPixelAt(int x,int y) {
        return this.bunky[x][y];
    }
```

```
    public void nulujPridano() {
        for (int i=0;i<velikost;i++) {
            for (int j=0;j<velikost;j++) {
                bunky[i][j].setPridano(false);
            }
        }
    }

    public double getCas() {
        return this.cas;
    }
}
```